Dipartimento di Informatica e
Scienze dell'Informazione

Design of Parallel Image Processing Libraries for Distributed and Grid Environments

by

Galizia Antonella

Theses Series **DISI-TH-2008-05**

DISI, Università di Genova
v. Dodecaneso 35, 16146 Genova, Italy http://www.disi.unige.it/

Università degli Studi di Genova

Dipartimento di Informatica e
Scienze dell'Informazione

Dottorato di Ricerca in Informatica

Ph.D. Thesis in Computer Science

Design of Parallel Image Processing Libraries for Distributed and Grid Environments

by

Galizia Antonella

February, 2008

Dottorato in Scienze e Tecnologie dell'Informazione e della Comunicazione
Dipartimento di Informatica e Scienze dell'Informazione
Università degli Studi di Genova

DISI, Univ. di Genova
via Dodecaneso 35, I-16146 Genova, Italy
http://www.disi.unige.it/

Ph.D. Thesis in Computer Science (S.S.D. INF/01)

Submitted by Antonella Galizia
DISI, Universitá di Genova
galizia@disi.unige.it

Date of submission: February 2008

Date of revision: June 2008

Title: Design of Parallel Image Processing Libraries for Distributed and Grid Environments

Advisor: Andrea Clematis
Istituto di Matematica Applicata e Tecnologie Informatiche
Consiglio Nazionale delle Ricerche, Genova, Italy
clematis@ge.imati.cnr.it

Supervisor: Vittoria Gianuzzi
DISI, Universitá di Genova
gianuzzi@disi.unige.it

Ext. Reviewers:

Julien Bourgeois
LIFC, Laboratoire d'Informatique de l'Université de Franche-Comté
Julien.Bourgeois@univ-fcomte.fr

Dieter Kranzlmuller
GUP, Institute of Graphics and Parallel Processing, Joh. Kepler University Linz
kranzlmueller@gup.jku.at

Beniamino Di Martino
DII, Seconda Universitá di Napoli
beniamino.dimartino@unina.it

Abstract

The general goal of this Thesis is the design of high performance tools for distributed image processing. Since the image processing demand of resources is increasing, parallel and distributed technologies represent an attractive solution for image processing computing intensive applications.

To achieve this goal we identify two main tasks: the first task is the development of a parallel image processing library, in order to ensure the satisfaction of high performance of requirements using available facilities such as a local parallel cluster. The second one is to enable the effective use of the library in different environments, i.e. distributed architectures and in particular the Grid. In this phase our scope is to plug the library in distributed frameworks taking into account the specific features of both environments.

To develop the image processing library, we consider the state of the art about the parallel image processing libraries, and we adopt some of the features well established in the specific field. Moreover, we combine them with topics not already considered in this application domain, as parallel I/O and the exploitation of advanced tools for data acquisition. A smart design and structuring of the library code is considered as well, as important examples in the linear algebra domain suggest. These features lead to the definition of a parallelization policy, which is applied by default during the executions of the library code. In order to provide a user-friendly tool, the parallelism is completely hidden to the users through the definition of a sequential interface. We consider a simple scheduling policy to enable load balancing when the applications are executed on heterogeneous resources.

A parallel library is mainly designed for a single cluster. In order to aggregate computational resources, and increase the library availability in distributed environments, we evolve it to new parallel distributed objects. In the implementation of such objects, we considered as key points: the library code reuse without a re-implementation or modification to it, the definition of user-friendly interfaces that still provide the library operations hiding their parallelism, the minimization of the overhead related with the exploitation of distributed environments, the preservation of the efficient parallelism of PIMA(GE)^2 Lib, and when it is possible, the exploitation of the scheduling policy applied in the library. To develop these tools we presented two approaches, the first addresses a client-server interaction in a general distributed environment, while the second studies the use of a parallel image processing library in a Grid environment. In both approaches, we simplified the exploitation of distributed and Grid environments since we provided services that hide the complexity of a direct use of the resources.

Actually, moving towards distributed architectures implies an overhead related with such environments, that is not possible to avoid, but only to reduce. It is added to the application performance, therefore we define further strategies to preserve the efficiency of the native library. We consider the applications as a pipeline of image processing operations that are applied on the input data set, and we analyse how to execute them in the most suitable way. The problem has to take into account the specific features of the underlying infrastructure. Thus, we investigate several points concerning distributed and Grid architectures, as for example the exploitation of parallel I/O architectures, the optimization of data management, the selection of the suitable Grid resources to execute the computations, and possibly the use of the scheduling policy of library. These topics have been carefully considered.

In each case, we supposed that the images to process could be stored on a remote host as well as the produced results, that could be loaded on a different remote destination. Hence remote data transfer were considered exploiting different protocols. The experimentations made with this aim converged in a I/O module, that enriches the I/O operations of PIMA(GE)^2 Lib.

We verify the effectiveness of our study through an example of real application in the field of Bioinformatics considering the analysis of Tissue MicroArray (TMA) images We have developed a Grid framework aimed to the TMA images handling through a Web graphical interface. TMAinspect allows the selection of TMA images through a query on metadata, the selection of the analysis to perform, and the monitoring of the execution on Grid resources.

The original contribution of the Thesis includes the following points:

- the definition of an efficient parallelization policy for image processing operations, a simple scheduling policy, and the development of a parallel library;
- the study about the issues related to the evolution of a parallel library into new high performance distributed objects, the definition of a strategy to enable the effective porting of a parallel library in distributed and Grid environments;
- the development of tools to enable the executions of high performance image processing applications in distributed and Grid environments;
- the exploitation of the scheduling policy of the library in a Grid environment;
- the development of an I/O module to acquire local and remote data;
- an application to real problems in the field of Bioinformatics, and the development of a related Grid framework.

To Naples, and Genoa
To my past, and my future
To my family, to Davide

So let us then try to climb the mountain,
not by stepping on what is below us,
but to pull us up at what is above us

(M.C.Escher)

Acknowledgements

I owe my first heartfelt thanks to Andrea, who believed in me and gave me the chance to work on what I most hankered. Thank you for your precious teaching, which guides me in the world of research; thank you for your understanding, support and encouragement. Thank you from the bottom of my heart for all you have done for me, Andrea dear.

Next comes Vittoria, who has coached me while still allowing me the utmost freedom of timing and choice. I mostly wish to thank you for the genuine interest shown when in trouble.

I would then like to thank Daniele, the saint, for many reasons: for your vital help at work, for sharing your knowledge with me, for your teaching at university, but mostly for putting up with me every day... I spend more time with you than I do with the rest of the world! Thank you for your support on the good days and on the not-so-good ones.

Thank you to me friend and co-author Federica, who allowed me to broaden the applications of my research to extremely interesting fields. Thank you Fede: I thoroughly like the work we develop together.

Then, there are my other dear colleagues: I thank them for the lunch breaks, chats and friendship offered to me right from my arrival in this town. It is from you that I got my first Genoese godchildren!

Again, thank you to me dear friend Massi, who opened up his life to me, thus allowing me to be part of something when everything around seemed to be foreign. Thank you, Massi, you are one of my strongest beacons. A thousand thanks for taking me to the Gradinata Nord!

Thank you to Genoa C.F.C., to red and blue Genoa which fills my life with emotions. Thanks to the UCGC which allows me to help those in need by spreading such feelings.

Thank you to all those in this town who are dear and near... Thank you, Genoa, for offering me the chance of a new life and yet another challenge. I have felt at home in Genoa, met many people, discovered new aspects of life... and met Davide.

Thank you to my family, who has allowed me to pursue my dreams and ambitions... I know full well how painful it is to miss someone every day. Thank you for your support and for the freedom of choice that you have given me. Thank you for your silent renouncing and for not burdening me with it. I hope to make you proud through my hard work, efforts

and sacrifices. Thank you to my grandmother and my nieces and nephews: my heart aches when I think of you, see you or have to say good-bye. I constantly miss you and this is something one can never get used to...

Thank you to Naples, my home town, to the warmth, folklore and liveliness that I so badly miss. Thank you to those who remain there, to all those with whom I am no longer in touch but who are in my heart, and to all the friends with whom I am still in touch today. Thank you to my old life...

Thank you to Marco, who has unwaveringly believed in me and backed up my choices, even when this implied making sacrifices. Thank you for the pride in your eyes: it has so often helped me through. I apologize if such choices have been hard to endure.

Thank you to Davide, the man I love, whose love brightens me and whose distance saddens me. A heartfelt thank you for your love, for the words you invariably have in store for me and for the serenity and happiness I find in your arms. You take care of me with your smile and wonderful eyes. Thank you, my love: I hope you will continue to do so for a long time!

And finally, a thank you to myself for struggling so hard to reach this far with my own strength...I am grateful to myself for sacrifices and choices I have had to make, though they are still, at times, somewhat difficult to endure. I would like to acknowledge the fact by challenging myself I have changed my life through sacrifices and the will to grow up. Thank you to myself for managing to achieve so much, making so many friends and finding so many interests though starting from scratch. Again, I should thank myself for allowing me to follow my dream and to carry out this project - of which my Ph.D. is part!

Ringraziamenti

E finalmente siamo arrivati ai ringraziamenti!

Il primo doveroso e sentito ringraziamento va ad Andrea, che ha creduto in me e mi ha dato la possibilitá di fare il lavoro a cui ambivo. Grazie per i preziosi insegnamenti che mi fanno da guida nel mondo della ricerca, grazie per la pazienza, il sostegno, e l'incoraggiamento. Grazie davvero di cuore per tutto quello che hai fatto per me, caro Andrea.
E subito a fianco Vittoria, che mi ha seguito lasciandomi la massima libertá, rispettando i miei tempi e le mie scelte. Soprattutto la ringrazio per il sincero interessamento dimostratomi quando mi ha visto in difficoltá.

Poi c'é quel sant'uomo di Daniele, e per lui i ringraziamenti sono molteplici: per la sua fondamentale collaborazione nel lavoro, per le cose che mi insegna, per la didattica, ma soprattutto perché mi sopporta tutti i giorni... passo piú tempo con lui che con il resto del mondo!grazie per il sostegno nei giorni normali ed in quelli un pó meno.
Un grazie alla mia amica e coautrice Federica, che mi ha dato la possibilitá di aprire le mie ricerche a settori molto interessanti. Grazie Fede, mi piacciono un sacco i lavori che sviluppiamo insieme.
Poi ci sono i miei cari colleghi, che ringrazio per le pause pranzo, le chiacchiere e l'amicizia che mi hanno offerto fin dal mio arrivo in questa cittá. É da loro che sono arrivati i miei primi nipotini genovesi!

Un grazie al mio caro amico Massi, che mi ha aperto la sua vita facendomi sentire di far parte di qualcosa quando tutto qui mi sembrava estraneo!Grazie Massi, sei uno dei miei punti di riferimento piú forti!E poi mille grazie perché mi hai portato in gradinata Nord! Grazie al Genoa, alla Genova rosso blu e mi riempie la vita di emozioni!Grazie ad UCGC che mi permette di aiutare il prossimo diffondendo queste stesse emozioni.

Grazie a tutte le persone che mi sono care in questa cittá... grazie a Genova che mi ha offerto la possibilitá di confrontarmi con una nuova vita e mi ha proposto un'altra sfida. Genova che mi ha fatto sentire a casa e che mi ha fatto conoscere tante cose e persone, e mi ha fatto incontrare Davide.

Grazie alla mia famiglia, che mi ha lasciato inseguire il mio sogno e le mie ambizioni... so quanto é doloroso tutti i giorni sentire la mancanza. Grazie per il sostegno e la libertá di scelta che mi avete sempre lasciato. Grazie per esservi privati in silenzio e senza mai farmelo pesare. Spero di rendervi orgogliosi con i miei impegni, sforzi e sacrifici. Grazie

alla mia nonna ed i miei nipotini, mi si stringe sempre il cuore quando vi penso, vedo o saluto. Mi mancate sempre, non si ci abitua mai...

Grazie alla mia Napoli, al suo calore e folklore, alla sua vitalitá che mi manca tanto, e tutte le persone che ho lasciato la... grazie a tutti quegli amici che ho perso nel tempo ma che sono sempre nel mio cuore, e grazie a tutti gli amici che continuo a sentire. Grazie a tutta la mia vecchia vita...
Grazie a Marco, che ha sempre creduto in me e mi ha fortemente incoraggiato nelle mie scelte, anche quando queste significavano sacrifici. Grazie per la fierezza che leggevo nei tuoi occhi, mi é servita in tanti momenti, mi dispiace se queste scelte sono state dure.

Grazie a Davide, l'uomo che amo!Davide che mi illumina con il suo amore, e mi rende triste con la sua distanza. A lui un grazie di cuore per il suo amore, per le parole dolci che trova per me e per la serenitá e la felicitá che sento tra le sue braccia. Da tempo ti prendi cura di me con il tuo sorriso ed i tuoi bellissimi occhi, grazie di cuore amore, spero lo farai ancora per molto!

Ed alla fine un grazie a me stessa, che ho lottato duramente per arrivare fin qua, con le mie sole forze... e ce ne sono volute tante. Grazie a me, per tutti i sacrifici e le scelte che dovuto fare, e per quanto siano ancora difficili in alcuni momenti. Grazie a me per essermi messa in discussione ed aver cambiato la mia vita, con sacrifici e voglia di crescere. Grazie a me, che sono riuscita a costruire tante cose, tante amicizie e tanti interessi partendo da zero. Grazie a me per essermi data la possibilitá di seguire i mio sogno, fare questo lavoro... e questo dottorato ne fa parte!

Contents

Chapter 1

Introduction

1.1 General aspects

Image processing is a topic of interest for a broad scientific community since a growing number of complex and considerable applications requires an adequate processing of images. Technological evolution permits large data acquisition through the use of sophisticated instruments, while scientific contributions enable the development of complex multidisciplinary applications to process them. In this way problems involving images have grown and nowadays cover several important scientific sectors, including medicine, industry, military, meteorology and earth observation.

However capturing, processing, and managing images to extract useful information need an enormous computational effort, often too high for a single processor architecture. The demand for computing resources makes necessary to move towards parallel and distributed computing. The emerging technologies to overcome these requirements lead to distributed architectures and Computational Grids, [10, 52]; both represent a target framework of our work, and they are described in the next Section.

As computational power advances, computational tools have to advance too. Our aim is to support the users to exploit the next computing architectures, without being aware of their implementation. Therefore considering the image processing community requirements, we would like to develop efficient and effective software that could provide useful tools for image processing scientists and contribute to the design of objects that could be considered as the evolution of library in distributed environments and Grids. To achieve this goal our research follows two directions:

- to develop a parallel image processing library, providing an interface that makes transparent parallel implementation. The library has to ensure high performance

and satisfy requirements of efficient processing in a network-based environment;

- to enable the use of the library in different environments. At this purpose, our scope is to merge the library in distributed frameworks ensuring the access to its functions through a simple interface.

High Performance and user friendliness are considered as fundamental requirements all over this work. During the Thesis we detail the major issues arising in the development of such parallel objects, and the strategies we implement to solve the critical points.

In this Chapter we introduce several topics concerning the Thesis; in the next Section we describe the general scenario of the work; it is composed by the domain application: image processing; the architectural domain: cluster, distributed environments, Grid; and software development domain: MPI, CORBA, Globus, gLite. In Section 1.3 we analyze the requirements of scientific community, they act as guide during the development of the Thesis. Section 1.4 provides a summary of the goals achieved during the Ph.D. research, and an outline of the Thesis.

1.2 The scenario

To better understand the difficulties arising during the Thesis, it is useful to define the context of our work, and outline the distinct aspects that represent the most relevant issues we have to address. We discuss elements regarding three main fields of interest, which are:

- the application domain, image processing;
- the architectural domain, parallel, distributed and Grid environments;
- the software development domain, parallel and distributed programming tools and environments.

This section provides a short survey about these topics.

1.2.1 Image Processing

Thanks to the new digital technology, it is now possible to manipulate multi-dimensional signals. Different subjects started depending on the kind of elaboration; they mainly could be divided into three categories [106]:

- Image Processing: when in the manipulation both input and output data are images. This kind of elaboration has a local nature, and the computation is performed applying an operation for each pixel in an image. Examples are: basic operations (e.g. point arithmetic, binary operations), basic filter operations (e.g., smoothing, edge enhancement), and image transformations (e.g., rotation, scaling).

- Image Analysis: when starting from an image, the elaboration extracts out measurements. In this case the image is reduced into segments (regions of interest), and the manipulation produces a more compact structure to represent the image. Examples are: region labeling, and object tracking.

- Image Understanding: when the manipulation starts with an image and extracts high-level description. It primarily concerns the output obtained from the Image Analysis, the operations try to imitate human cognition and decision making. Examples are: object recognition, and semantic scene interpretation.

Since the three categories are built one upon the other, they are also called: low level, intermediate level, and high level image processing.

This work is focused on *image processing*, or *low level image processing*; usually images could be photographs or frames of video or images acquired using specific instruments such as satellites, biomedical machines, and more in general imaging instruments. It operates directly on an image, processing the individual pixel values to modify the image in some way. The processing of visual data at the pixel level has a highly regular nature; it provides a natural source of parallelism.

In order to provide useful tool in the development of image processing application, we considered the *Image Algebra* theory [106] as the base of this work. It was born to satisfy the need for precision and concision in image processing and computer vision, and provides a coherent algebra which can be employed to solve problems in such fields. Starting from the Image Algebra theory, it is possible to outline a set of operations that represents a common kernel for many typical image processing applications. Considering that in complex image processing applications, a significant amount of execution time may be spent in these low-level operations, reducing their execution time leads to an improvement of performance of the programs. As a consequence, to provide an efficient implementation of low level operations is a crucial element to obtain high performance executions.

1.2.2 The Architectural Domain

Since the early 1980's there has been an increasing interest of the scientific community for parallel computing and distributed systems.

Indeed parallel computers became available in a variety of architectures, and also the networking hardware and software were becoming more widely deployed. Thus it was necessary to develop parallel programming languages and tools in order to effectively use these parallel machines [122, 45]. Exploiting computer networks, it was natural to group machines together and use them for the executions of parallel codes. Besides homogeneous machines, sets of heterogeneous machines have been considered for the executions of parallel codes, [84, 67]. In this way the notion of distributed computing was starting, and distributed computing tools were developed, such as Distributed Computing Environment (DCE) [97], and Common Object Request Broker Architecture (CORBA) [33]. Therefore the same application could be subdivided in several execution threads running on different machines. In this way the concept of *metacomputing* was born [125]; this word has been used to describe different resources, computing and instruments, geographically distributed, used as a single powerful parallel machine. Scientific community was more and more interested in distributed computing, since it understood the potentialities of distributed supercomputing. Step by step a tremendous amount of research was done to overcome the intrinsic obstacles of distributed architectures and to perform robust computations ensuring high performance. More and more possibilities were guessed and requirements were achieved in different projects, such as I-WAY [36], Nexus [50]. The definition of Grid was starting [128], and projects such as Globus [61], Legion [68], Condor [49] have proved its feasibility. By 1998 *Grid computing* was firmly established with the publication of the book by Foster and Kesselman [52], in its first edition.

Thereafter an overview of the architectural environments we already introduced is provided.

1.2.2.1 Cluster based parallel computing

Since 1990s there has been an increasing trend to move away from expensive and specialized proprietary parallel supercomputers towards networks of workstations. One of the causes is the rapid improvement in the availability of high performance components for workstations and networks. Clusters can easily grow, as node's capability can be easily increased by adding memory or additional processors. This made networks of computers (single or multiprocessor systems) an appealing vehicle to obtain parallel low-cost processing.

In order to give a precise definition [12], we can say that a *commodity cluster* is a local computing system comprising a set of independent computers and a network interconnecting them. A cluster is *local* when all its component subsystems are supervised within a single administrative domain, and it usually resides in a single room and it is managed as a single computer system. The *computer nodes* are commercial-off-the-shelf (COTS) elements, capable of full independent operation and ordinarily employed individually for mainstream work loads and applications. The nodes may incorporate a single microprocessor or multiple microprocessors in symmetric multiprocessor (SPM) configuration. The

interconnection network employs COTS local area network (LAN), it is dedicated to the integration of the computing nodes and is separate from the external environment.

A cluster may be employed in many modes including but not limited to: high capability or sustained performance on a single problem, high capacity or throughput on a job or process workload, high availability through redundancy of nodes, or high bandwidth through multiplicity of disk and disk access or I/O channels.

1.2.2.2 Heterogeneous cluster collections

Nowadays it is often the case that more than one cluster is available at one location. Each cluster is normally homogeneous but the collection of clusters represents an heterogeneous computing resource.

The availability of high speed networks, that connect the different clusters, makes this collection of computing resources amenable for parallel applications. This is true in particular for complex applications that put together in a pipeline assembly different software components that are internally parallel.

1.2.2.3 Distributed computing

A distributed environment is a collection of individual computing devices that can communicate with each other. This very general definition includes a wide range of modern computer systems, such as homogeneous local clusters, workstations, parallel machines. Distributed computing brings in computers owned by the general public over the world. Popular projects based on distributed home commodity facilities include SETI@home [117], distributed.net [39], GIMPS [64] and many others.

Although distributed computing systems are highly desirable, putting together an adequate software system is notoriously difficult. In fact despite the availability of hardware for distributed systems, there are few software applications that are able to exploit the hardware. One of the simplest reasons is that distributed software requires a different set of tools and techniques than the ones required by the traditional software. A pragmatic difficulty is for example represented by the presence of heterogeneous hardware and operating systems. It leads to an inevitable lack of adherence to standard. More in general three factors represent the most important obstacles in the development of software:

- absence of a shared clock: in a distributed system it is impossible to synchronize the clocks of different processors precisely because of the uncertainly in communication delays between the processors. For this reason the system is asynchronous: since the time at which an event takes place can not always be known precisely;

- absence of a shared memory: in a distributed system it is impossible for each processor to know the global state of the system. As result, it is difficult to observe any global property of the system. For this reason the system has a local knowledge: since each computing element can only be aware of the information that it acquires, it has only a local view of the global situation;

- absence of accurate failure detection: in a distributed system it is impossible for any processor to distinguish between a slow process and a failed one. For this reason the system is very sensitive to the failures: computing elements can fail independently, leaving some components pending.

There is not a single universally accepted model of computations in a distributed environment, they mainly differ in the management of the previous points, i.e. how computing elements communicate, the timing information and behavior available, and the tolerated failures [133]. Examples of paradigms for distributed systems are: Client-server, 3-tier architecture, N-tier architecture, Distributed objects, loose coupling, tight coupling, Peer-to-peer [142].

1.2.2.4 Grid Computing

A Grid is a persistent environment that enables software applications to integrate and use instruments, displays, computational and information resources that are managed by various organizations in widespread locations. Sharing of Grid resources is obtained through a direct access to computers, software, data, and devices, as it is required by a range of collaborative problem-solving and resource-brokering strategies emerging in science, but also in industry, engineering, and services. This sharing must be highly controlled, with a clear definition of what is shared, who is allowed to share, and the conditions under which sharing occurs. A set of individuals and/or institutions defined by such sharing rules forms a *Virtual Organization* (VO) [53]. The *Open Grid Services Architecture* (OGSA) [54] provides an extensible set of services that VOs can aggregate in various ways. Every resource is treated as a *service*-entity that provides some capability through the exchange of messages. In this way OGSA defines a *Grid service*, that is a Web service providing a set of well-defined interfaces and following specific conventions; adopting this uniform services-oriented model makes all components of the environment virtual.

The current Grid activity is regulated by the Open Grid Forum (OGF) [95], it is the hub for the world-wide grid community of computational scientists, industrial partners, physicists, and engineers. The OGF seeks to ratify community standards for Grid software and services; it coordinates a growing number of groups, over 400 organizations in more than 50 countries, each focused around a different topics of grid computing. OGF is the

"new" organization that resulted from the merge of the Global Grid Forum (GGF) and the Enterprise Grid Alliance (EGA).

Nowadays there is a clear separation between Computational Grid and Data Grid. Computational Grids provide the basic services and protocols to manage general Grid resources. They enable user's development of further Grid services to manage resources, such services are tuned considering the user requirements. Data Grids are mainly aimed to the development of vertical applications in specific application domains. Thus, they already provide the services defined in OGSA, and they are oriented to the analysis of data rather then the development of further services to manage Grid resources. During the Thesis we consider the use of both Grids.

Portals

It is very difficult to find a consistent definition of what exactly is a portal. Here we consider a portal as an entity that provides information and services for a particular user community or topic of interest through a combination of site information and links. Examples of well known portals are: Google, Yahoo, Excite, etc. A Grid portal is a user's point of access to a Grid system. It provides an environment where the user can access Grid resources and services, execute and monitoring Grid applications, and collaborate with other users. A number of Grid portals exist including the NEESgrid Portal [92], the GENIUS Portal [14], the Alliance Portal [91]. Also several large collaborative efforts to build portal construction toolkit based on OGSA Grid Services are available, such as GridSphere [66].

The architectures cited above are based on the idea that a portal is a container for "user facing" Grid services which are signed according to the portlet model. A portlet is a server component that controls a small, user-configured window into the user's web browser.

Nowadays many Grid projects, mainly concerning Data Grids and specific domain applications, are organized as portals that provide suitable services to enable the analysis of shared data ensuring secure data access. They hide the burdens related to the exploitation of Grid resources and allow to develop new applications, examples in Bioinformatics are provided by [120, 119].

1.2.3 Software development domain

Current parallel computing platforms are evolving from homogeneous single cluster to collection of heterogeneous clusters. According with this evolution in the field of parallel programming there is a need to evolve from the use of message passing libraries to higher level programming languages and to produce new programming tools to develop performance portable programs over different architectures. Moreover it is often necessary to build complex applications as the composition of internally parallel software components.

In the following subsection we exemplify the programming tools commonly employed in such situations, MPI, a message passing library, CORBA a framework for distributed environment, Grid middleware such as Globus and gLite.

1.2.3.1 MPI

The Message Passing Interface (MPI) defines a standard API that is widely used in parallel computing. It results as the amalgamation of what were considered the best aspects of the most popular message passing systems at the time of its conception. This work was undertaken by the MPI Forum [90], a committee composed of vendors and users formed at the Supercomputing '92 Conference with the aim of defining a message passing standard. The goals of the MPI design were portability, and efficiency. The standard defines a message passing library and leaves, among other things, the initialization and control of processes to be defined by individual developers. MPI bindings are available for Fortran 77, Fortran 90, ANSI C and C++ on a wide range of platforms from tightly coupled systems to metacomputers. MPI became the "de facto" standard for message passing.

The computing model is based on the notion that an application consists of several *tasks*. Each task is responsible for a part of the application computational workload. Sometimes an application is parallelized along its functions: this process is often called *functional parallelism*. A more common method of parallelizing an application is called *data parallelism*: all the tasks are the same, but each one knows and treats only a small part of the data. This is also referred to as the *Single-Program Multiple-Data* (SPMD) model of computing. MPI supports either or a mixture of these methods. Depending on their functions, tasks may execute in parallel and may need to synchronize or exchange data.

An extension of the MPI1 was made in the 1997, when the MPI Forum defined MPI2. It includes all the MPI1 specifications, but some features are considered obsolete, and their use is deprecated. The major extensions are: Parallel I/O, Remote Memory Operations, and Dynamic Process Management. Moreover MPI2 supports the interaction with threads, and enables interoperability among different languages.

The MPI2 *Parallel I/O* provides a high-level interface for the partition of data file among processes and a collective interface for the complete transfer of global data structures between process memories and files. It also supports asynchronous I/O, strided accesses, and controls over physical file layout on storage devices (disks). We have utilized this efficient system during the Thesis, see Chapter 3.

Remote Memory Access (RMA) extends the communication mechanisms of MPI by allowing one process to specify all communication parameters, both for the sending side and for the receiving side. Data in a fixed range is made available by one process, and they can be accessed from the others. RMA works as a shared memory.

Dynamic Process Management allows the creation and termination of processes after an MPI application has started. It provides a mechanism to establish communication between the newly created processes and the existing MPI application.

1.2.3.2 Middleware for distributed environments: CORBA

The problem of resources integration in a distributed environment through the use of suitable middleware has been considered before and out of the Grid community. These efforts lead to different results that now turn out to be of interest. In particular we consider the Common Object Request Broker Architecture [137], that is a specification from Object Management Group (OMG) [98] to support distributed object-oriented applications. It provides a framework where heterogeneous distributed objects are integrated and fused within one system, without modifying the native code.

The effective communication channel is a part of the framework called Object Request Broker (ORB), which allows the interacting objects to locate each other; it also provides network data transport and component communications in a distributed domain. The components are seen as black blocks enabled to perform specific services, explicated by their interface through the CORBA Interface Definition Language (IDL). This is a declarative language with the important feature to be language independent, therefore it uniforms the objects, despite they use different programming language or hardware and software platforms. The objects specify their services through the IDL, and ORB delivers request from a client to a server; obviously no constraints are given on how the objects have to satisfy a request.

The last CORBA specification extends the CORBA Object Model (COM) to the CORBA Component Model (CCM) [140]. From CORBA 3.0, CCM is used to standardize the component development cycle using CORBA as its middleware infrastructure. This extension permits to implement, manage, configure and develop components and the CCM becomes the definition of features and services within a standard component oriented environment. Further details about CORBA are provided in Chapter 5.

1.2.3.3 Grid Middleware

Many difficulties arise in the context of Grid computing, for example to enable flexible, secure and coordinated sharing and coupling of distributed heterogeneous resources. The user community strongly recognizes the need for advanced environments, which support a suitable level of abstraction to develop complex high-performance applications and to preserve, in the meanwhile, properties such as reliability and performance. Such environments should also allow users to easy integrate software tools required to develop a large

and complex application.

Other difficulties are related with High Performance, for example in order to get better performance, the challenge is to map each software component onto the best candidate computational resource having high degree of affinity with the software component; this kind of mapping is a non-trivial task. Furthermore an efficient execution of parallel applications is a complex task in the Grid environment, since the effective selection of the resources that are able to provide the highest performance is complex to achieve. The major issue is to limit the data movement among the repositories, the computational resources and the users using low bandwidth links. Another point is to consider machines belonging to the same LAN, the communication time during the computation is lower than for machines belonging to different organizations. Another advantage is represented by the presence of advanced parallel tools for data acquisition.

Today, different solutions have been proposed to develop and execute efficient applications, here we present the basic middlewares of a Computational Grid and of a Data Grid to introduce the tools we employ in the Thesis, see Chapter 6.

Globus

The "de facto standard" for Grid computing is the Globus Toolkit [51]. It represents a community-based, open-source set of Grid services and libraries through which applications can handle distributed and heterogeneous resources as a single virtual machine.

The toolkit addresses issues of security, information discovery, resource management, data transfer, communication, and portability. Globus Toolkit mechanisms are used by dozen of major Grid projects worldwide. The Globus Alliance, [61], is a research and software development project coordinated by Ian Foster and Carl Kesselman, generally considered the "fathers" of Grid Computing.

The Globus Toolkit latest version, GT4, is the first full-scale implementation of the Open Grid Services Architecture (OGSA) specifications. It adds to the previous version, GT3, other components and tools to support a successful Grid environment. It implements many protocols defined by OGF. The previous generation toolkit, GT2, is still supported as user community makes the transition to Grid services, but its use is defined deprecated.

gLite

The gLite middleware is implemented by the EGEE project,[60]. gLite provides foundation services that facilitate the building of Grid applications from all fields. In fact gLite is mainly aimed to enable the analysis of data, and to be used by less expert Grid users. Thus gLite represents a modular system where users could deploy different services according to their needs, thus tailoring the system to their individual situation.

The gLite Grid services follow a Service Oriented Architecture, thus it ensures easy connec-

tion of the software to other Grid services, and compliance with upcoming Grid standards, as the OGSA from the OGF. Actually gLite integrates components from successful middleware projects, such as Condor and the Globus Toolkit, as well as components developed for the LCG project, LCH (LCH Large Hadron Collider project) Computing Grid, and it is compatible with schedulers such as PBS, the Portable Batch System [2], Condor [32] and LSF, Load Sharing Facility [104].

As already said, gLite is mainly aimed to enable less expert users to exploit the Grid potentialities in the analysis of data, thus it has improved features as security, interfaces for data management and job submission, a re-factored information system, and others to make gLite easy to use.

1.3 Requirements and goals

In this Section we provide an overview of the requirements that acted as guide in the development of the Thesis. They are related with the different architectures we exploit during our research.

We started with a simple and architecture-independent consideration, even if image processing applications need computing power and efficient executions, the imaging community is not so expert in achieving these goals. Therefore it prefers to employ such tools and environments that shields from the burdens of using high performance architectures.

Thus, it is essential to provide tools that offer a more *familiar* programming model. The development of user friendly tools, able to ensure high performance executions, is mandatory. The best way to achieve this goal is to develop tools that hide the parallelism of the computation, produce efficient code in most situations, and manage as transparently as possible the underlying infrastructures. For each considered architecture, we look for a proper and efficient model to provide image processing operations. Each solution will be discussed in the suitable Chapter, however it is necessary to take into account a list of general requirements that could be refined in the different architectures.

1.3.1 Homogeneous parallel machines

Starting with the development of parallel image processing library, mainly designed local clusters, we can consider as basic requirements the following:

- easiness of use, the most compelling need for the users is a simple and intuitive tool to exploit the provided operations, therefore a good point is represented by a well designed interface;

- efficiency, the users should be capable of obtaining significant performance gains in the most common image processing operations, when they are executed as single operation and composed in a pipeline;
- opaque interface hiding the parallelism and the optimization policy, it is essential to enable the user's exploitation of an effective parallelization strategy without managing the intrinsic complexities of such computations;
- modularity, it is important to take care during the implementation of the library to organize as better as possible the code, to avoid unnecessary code redundancy, and to enhance operation reusability;
- extensibility, the user has to be able to insert new operations to the library code, this step should not require any parallelization effort to the user and changes to the original source code;
- portability, it is essential to ensure correct executions on different target machines, to provide computations that are insensitive to the variations of the data and ensure correct results. To implement code with an high-level language is a good choice;
- scalability, a well-designed software architecture has to increase its performance when computational resources are added to execute the elaboration, or the load of the problem increases;

In order to improve performance and the portability of the library during our research, we would like to provide an extension of this list adding other important requirements to achieve:

- parallel I/O, in order to improve the library efficiency a good way is to perform the Input/Output operations in parallel, to exploit the underlying file system and to have an effective use of its functionalities;
- memory management, the exploitation of memory hierarchy or the cache memory size is a key point to enable an efficient memory management and the improvement of the performance;
- load balancing, considering heterogeneous architectures it is necessary to consider an effective policy that takes into account the computational power of the nodes and thus avoids possible load unbalancing during the computation;

1.3.2 Distributed environments and Grids

Moving towards distributed and grid environments, it is necessary to consider also other important issues. A parallel image processing library in a distributed environment has to ensure:

- interoperability, the code should be able to exchange data and information with other software components, in order to be employed in multidisciplinary applications;
- high performance, the code should be capable of obtaining significant performance gains, especially for operations that are most commonly applied in image processing research, also in complex environments;
- virtualization of the underlying infrastructures, the code should make easier the exploitation of distributed infrastructures, and not expose users to its complexities;

Grid environments are much more complex than the parallel and distributed ones. Therefore to exploit a grid, we have to consider additional requests:

- abstraction, since the Grid is a complex environment, it is important to abstract from the unnecessary information related with infrastructure, for example precise details of computational hardware, or replication of processing units;
- aggregation, a code has to be aware of the capabilities present on the grid and to be able to aggregate them. Both the aggregation of capacity, i.e. clustering computational power or storage elements, as well as aggregation of capability, i.e. combining a specialized instrument with storage elements, are needed;
- efficient execution, it is useful define a strategy to map each software component onto the best candidate computational resource to ensure better performance, thus a proper exploitation of Grid resources has to be ensured;
- dynamicity, the grid is intrinsically dynamic, and this concept involves both the hardware capacity, both the knowledge shared on a grid. It is necessary to provide a minimum level of support for the dynamism of the grid;
- multi domain, resources in a grid belong to many different organizations, with an different security and administrative polices. It is essential to define virtual organizations to access virtual environments for resource sharing, on line interaction and collaborations;
- security, secure access to resources is an essential feature of a grid; therefore authorized users are allowed to use specific resources, through a limited number of operations. It is underlined multi administrative domains.

1.4 A summary of the achieved results

The subject of the Thesis is the design of high performance tools for distributed image processing. In particular we deal with the issue of the development of a parallel image processing library and enabling its efficient use in distributed environments and Grids. For this reason, the first step has been:

- the development of a parallel image processing library, called PIMA(GE)^2 Lib, Parallel IMAGE processing GEnoa Library, to obtain high performance executions on local resources such as a cluster;

PIMA(GE)^2 Lib is based on an efficient parallelization policy, and provides the commonly used image processing operations. The library enables the elaboration of 2D and 3D data sets, and shields the users from the burdens of parallel computations through a sequential interface. PIMA(GE)^2 Lib exploits a parallel I/O and advanced parallel tools for data acquisition and also apply a policy to ensure the load balancing of the executions on heterogeneous architectures.

A parallel library is mainly designed for a single local cluster. In order to aggregate computational resources, and increase the library availability in distributed environments, we evolve it to new parallel distributed entities. We have a specific requirement to satisfy: an interoperable use of the PIMA(GE)^2 Lib operations. Typically multidisciplinary applications are composed of large number of complex components; some of them may involve images and require an efficient processing. We would like to enable the use of the library as the building block to develop applications in these cases. Furthermore it is necessary to exploit dedicated resources to enable efficient executions of parallel applications.

In the implementation of the parallel distributed entities we considered as key points: the library code reuse without a re-implementation or modification to it, the definition of user-friendly tools that still provide the library operations hiding their parallelism, the preservation of the efficient parallelism of PIMA(GE)^2 Lib, and when it was possible, the exploitation of the scheduling policy applied in PIMA(GE)^2 Lib.

However moving towards distributed architectures implies an overhead related with such environments, which is not possible to avoid, but only to minimize. It is added to the application performance, therefore another critical point is to preserve good speed up values through the definition of further optimization strategies.

To achieve this goal we investigate several points in the construction of objects for high performance distributed image processing. Applications have been seen as a pipeline of image processing operations applied on the input data set; therefore it is necessary to execute them in the most suitable and efficient way. The problem has to take into account the specific features of the underlying infrastructure. We investigate several points concerning

with distributed and Grid architectures, as for example the exploitation of advanced tools for data acquisition, an efficient data management, the selection of the suitable Grid resources to execute the computations, and the possible exploitation of the scheduling policy of the library. These topics have been carefully considered.

In the Thesis we present two approaches aimed to this general goal. The first approach addresses a client-server interaction in a general distributed environment, while the second approach studies the use of a parallel image processing library in a Grid environment.

In the first case the server provides services that correspond to the execution of library functionalities, possibly at different level of granularity. The server is statically bounded to a fixed set of computing resources, such as a cluster, and is remotely located from its clients. No specific infrastructure is assumed to exist in this case, but we rely on frameworks that enable interoperability in distributed environments, such as CORBA.

In the Grid environment we assume the existence of a middleware that implements a distributed virtual infrastructure. In this case it is possible to foresee different situations to enable the use of a parallel library. A remarkable characteristic is the availability of dynamic computing resources.

In both distributed objects, each application request is executed on a single parallel machine, since we did not address the topic of managing the same parallel execution on different machines. This is a non-trivial problem, which requires the implementation of further strategies in order to allow the execution. For the same reason, we also avoid the topic of rescheduling the application when anomalies or failures occur.

In each case, we supposed that the images to process could be stored on a remote host as well as the produced results, that could be loaded on a different remote destination. Hence remote data transfer were considered exploiting different protocols. The experimentations made with this aim have converged in a module for remote I/O, that enriches the efficient I/O operations of PIMA(GE)^2 Lib.

We verify the effectiveness of our study through an example of application to real problems. We consider the elaboration of images obtained through the Tissue MicroArray technique; in this context we performed several steps. The first step was the development of a set of operations through PIMA(GE)^2 Lib to outline useful information from the TMA images. The second was the exploitation of the Grid to enable data acquisition complying with a possible restricted access policy defined by the institutions that own TMA data. Other efforts were spent to enable queries on descriptive metadata of the TMA images, and thus to search interesting data shared on the Grid. Last step was the execution of the TMA image analysis on the Grid, preserving the parallelism of the operations and their sequential interface. These works constitute TMAinspect, a Grid framework aimed to the TMA images handling through a Web graphical interface.

The original contribution of the Thesis includes the following points:

- the definition of an efficient parallelization policy for image processing operations, a simple scheduling policy, and the development of a parallel library;
- the study about the issues related to the evolution of a parallel library into new high performance distributed objects, the definition of a strategy to enable the effective porting of a parallel library in distributed and Grid environments;
- the development of tools to enable the executions of high performance image processing applications in distributed environments;
- the exploitation of the scheduling policy of the library in a Grid environment;
- the development of an I/O module to acquire local and remote data;
- an application to real problems in the field of Bioinformatics, and the development of a related Grid framework.

We detail all these aspects during the remainder of Theses; it is possible to make available on request, the works presented in Chapter 3 and 5, and a part of the works presented in Chapter 6.

1.4.1 Projects and Publications

This Thesis was supported by the following projects:

- MIUR program L. 449/97-00 "High Performance Distributed Platform",
- FIRB strategic Project on Enabling Technologies for Information Society, "Grid.it";
- the regional program of innovative actions PRAI-FESR Liguria.

The works described in Chapter 1, 2 e 3 have been developed in the MIUR program; in the FIRB strategic project we developed the works presented in Chapter 5 and a part of the works presented in Chapter 6. The remaining of the goals of the Thesis have been obtained in the PRAI-FERS regional program, and in a collaboration with the ITB-CNR, Milan.

Most of the results that are presented in the Thesis are the subject of the following publications.

The description of the parallel I/O applied in the PIMA(GE)^2 Lib was presented in:

- Clematis A., D'Agostino D., Galizia A.
  Parallel I/O Aspects in PIMA(GE)^2 Lib.
  In: Parallel Computing: Architectures, Algorithms and Applications, Proceedings of the ParCo 2007, Advances in Parallel Computing, Volume 15, pp. 441-448, IOS Press, 2007.

Some of the aspects of PIMA(GE)^2 Server were presented in:

- Clematis A., D'Agostino D., Galizia A.
  An Object Interface for Interoperability of Image Processing Parallel Library in a Distributed Environment.
  In the Proceeding of ICIAP 2005, LNCS vol. 3617, pp. 584-591, Springer, 2005.

- Clematis A., D'Agostino D., Galizia A.
  A Parallel IMAGE Processing Server for Distributed Applications.
  In: Parallel Computing: Current & Future Issues of High-End Computing, Proceedings of ParCo 2005, pp. 607-614, NIC Series vol.33, John von Neumann Institute fuer Computing, 2006.

The first experiences in the analysis of TMA images on the Grid were presented in:

- Viti F., Merelli I., Galizia A., D'Agostino D., Clematis A., Milanesi L.
  Tissue MicroArray: a distributed Grid approach for image analysis.
  In: From Genes to Personalized HealthCare: Grid Solutions for the Life Sciences, Proceedings of HealthGrid 2007, Studies in HealthTechnology and Informatics, Vol. 126, pp. 291-308, IOS Press, 2007.

- Galizia A., Viti F., D'Agostino D., Merelli I., Milanesi L., Clematis A.
  Experimenting Grid Protocols to Improve Privacy Preservation on Efficient Image Processing.
  In: Parallel Computing: Architectures, Algorithms and Applications, Proceedings of the ParCo 2007, Advances in Parallel Computing, pp. 139-146, Volume 15, IOS Press, 2007.

The analysis presented in these papers have been published as *Image of the week* in the International Science Grid This Week [70], Issue 28, June 2007.

The development of Grid systems to enable a simplified exploitation of Grid resources were presented in:

- Galizia A., D'Agostino D., Clematis A.
  The use of PIMA(GE)^2 Lib for Efficient Image Processing in a Grid Environment.
  In the Proceedings of INGRID 2007, in press in a book edited by Springer.

- Galizia A., Viti F., Orro A., D'Agostino D., Merelli I., Milanesi L., Clematis A.
  Enabling parallel TMA image analysis in a Grid environment.
  In IEEE Proceedings of CISIS-2008, in press by IEEE Computer Society.

- Galizia A., Viti F., Orro A., D'Agostino D., Merelli I., Milanesi L., Clematis A.
  TMAinspect, an EGEE Framework for Tissue MicoroArray Image Handling. In IEEE Proceedings of CGRID 2008, in press by IEEE Computer Society.

Further results were described in different Technical Reports:

- Clematis A., D'Agostino D., Galizia A.
  The Parallel IMAGE processing GEnoa Library: PIMA(GE)^2 Lib.
  Technical Report IMATI-GE 21/2006, 2006.

- Galizia A., D'Agostino D., Clematis A.
  The use of PIMA(GE)^2 Library for Efficient Image Processing in a Grid Environment.
  Technical Report IMATI-GE 18/2006.

- Clematis A., D'Agostino D., Galizia A.
  Exploiting Parallel I/O for Image Processing.
  Technical Report IMATI-GE 14/2006, 2006.

- Galizia A., D'Agostino D., Clematis A.
  Un esempio di code encapsulation tramite CORBA.
  Technical Report IMATI-GE 21/2004, 2004.

- Galizia A.
  Valutazione di tecniche di ottimizzazione nell'implementazione di Librerie Parallele.
  Technical Report IMATI-GE, N. 20/2004.

- Galizia A., D'Agostino D., Clematis A.
  Integrazione e sviluppo di librerie parallele in un ambiente di programmazione Grid-aware basato su un modello a componenti.
  Technical Report IMATI-GE 19/2004, 2004.

- Galizia A.
  A ParHorus Overview.
  Technical Report IMATI-GE, N. 18/2004.

- Galizia A., Bottini N.
  Basic Building Block for an Image Processing Library.
  Technical Report IMATI-GE, N. 8/2003.

1.4.2 Thesis outline

In this Chapter we already give an overview of the Ph.D. research, we define the objectives, the scenario, and the goals of the Thesis.

Chapter 2 represents an introduction about the development of a parallel image processing library, and it discusses the well-know example of the numerical libraries and a relevant image processing library. Thus the Chapter describes numerical libraries and their evolution, from BLAS to distributed frameworks. We also analyze the characteristics that we would like to inherit in our library. Moving towards image processing, we analyze ParHorus, an interesting example of parallel library. Also in this case we outline the characteristics that we would like to inherit. The last section of the Chapter describes the additional features of our library.

Chapter 3 describes PIMA(GE)^2 Lib, Parallel processing IMAGE GEnoa Library. The library elaborates 2D and 3D data set applying data parallelism. The parallelism is hidden to the user through a sequential API, obtained by defining a hierarchy of objects-image processing operations. The parallelization policy is obtained considering appropriate patterns for the elaborations, an efficient parallel I/O, and a suitable policy to ensure load balancing on heterogeneous resources. The last Section introduces a real application, that we consider as a case study during the Thesis: the analysis of images obtained by the Tissue MicroArray technology.

With Chapter 4, we move toward distributed and Grid environments, our aim is to enable the use of the library in the development of distributed tools for parallel image processing. We have to evolve the concept of library into a more appropriate model. In particular we consider two types of interactions with the user: client-server architectures for a classic distributed environment, and service oriented approach for the Grid. We describe the general issues arising in the integration and we outline the possible solutions: an appropriate management of data, and the preservation of the granularity of applications.

Chapter 5 describes PIMA(GE)^2 Server; it has been developed using CORBA. The server hides to programmers complexity due to use of parallel architectures and environments the transparence, and it defines an effective policy to enable parallel computations in CORBA objects without modifying the standard defined by the OMG. PIMA(GE)^2 Server allows the access to remote data, their transfer with appropriate protocols, and it manages the computation of the applications preserving their granularity. Experimental results demonstrate the effectiveness of strategies implemented.

Chapter 6 describes the different experiences to support the use and the execution of the library on the Grid. In this direction, we have exploited the grids in order to achieve different goals: 1) we develop new services to allow the execution of applications developed with the native library. This experimentation leads to the definition of PIMA(GE)^2 Grid. 2) We

enable the use of specific Grid protocols in the library in order to extend its functionalities. We exploit gLite and Grid File Access Library, GFAL, to allow data access complying with a possible restricted access policy. 3) We consider the use of the library to develop vertical Grid applications oriented to specific application domains. Also in this case, we exploit gLite and we develop a framework, called TMAinspect, to perform the analysis of TMA images on the Grid.

Chapter 2

Preliminary remarks in the development of a parallel image processing library

To outline important issues in the design of parallel image processing library, we made a preliminary study about parallel libraries. We looked at two interesting examples: the ScaLAPACK and ParHorus libraries; they represent significant parallel packages, respectively among linear algebra and image processing tools. We deeply analyzed their organization and their features, in order to take into account these elements during the development of PIMA(GE)^2 Lib, Parallel IMAGE processing GEnoa Library.

Therefore this Chapter represents an introduction to the development of a software library, and is organized as follow: we start from the linear algebra; we describe the beginning and the evolution of the BLAS library [72, 43, 42], other libraries based on them such as LAPACK [5], the description of ScaLAPACK [16], the recent Self Adaptive Numerical Software project [40], and to conclude we briefly mention some projects aimed to the integration of such libraries in distributed and grid environments, such as Cactus [63], NetSolve/GridSolve [1], and VGrADS [136].

The overview about numerical libraries has to be considered as an arrangement phase, since the image processing library we propose could not reach a similar level of development considering the different dimension of the research activities under which it has been developed.

After that, we move toward the image processing and the tools that enable a simple implementation of high performance applications. We selected ParHorus [113]; it is analyzed in deep, explaining the notable points of that work. For both cases, we clearly underline the interesting features that we want to preserve during the development of PIMA(GE)^2

Lib as well as the further features that extend and characterize PIMA(GE)^2 Lib.

2.1 The numerical library model

Numerical linear algebra is often the heart of many engineering and computational science problems, such as image and signal processing, computational finance, materials science simulations, structural biology, data mining, and bioinformatics just to name a few [44]. Actually it has been estimated that the main bulk of about 70% of the problems posed in the scientific community admits as solution a numerical linear algebra algorithm. Thus the development of numerical linear algebra libraries had a great improvement at the end of the 70's. Those libraries provide the *building blocks* for the efficient implementation of more complex algorithms, such as the solution of linear systems of equations, linear least squares problems, eigenvalue problems and singular value problems. Since the development of efficient software has to take into account different key points, it is essential to make available an implementation of computational kernels optimized with respect to the underlying architectures. For this reason software libraries have been improved and evolved considering also the specific features of the architectures established during the years. For example an important aspect is an efficient memory management; in most computers, performance of algorithms can be dominated by the amount of memory traffic, rather than the number of floating-point operations involved. The movement of data between memory and registers can be as costly as arithmetic operations on the data. Thus a key issue is the exploitation of computers with a hierarchy of memory (such as global memory, cache or local memory, and vector registers) and true parallel-processing computers. These aspects lead to a restructuring of existing algorithms to enable performance improvement. In the remaining part of the Section, we describe this topic, through the description of different famous packages.

2.1.1 Basic Linear Algebra Subprograms

BLAS, as it is commonly called, is a very successful example of software library and is used in a wide range of software. It is an aid to clarity, portability, modularity, and maintenance of software; it becomes a *de facto* standard for elementary vector operations.

BLAS identifies the frequently occurring operations of linear algebra, i.e. the *building blocks*, and specifies a standard interface for them, thus promoting software modularity. To improve performance, the optimization of BLAS subroutines can be done without modifying the higher-level code that may employ them. Other peculiar features of BLAS code are robustness, portability and readability. It is possible to identify three Levels of BLAS, depending on the software organization; this structure is aimed to obtain a better

exploitation of the underlying architecture and to improve performance.

2.1.1.1 Level 1 BLAS

The first major effort in the development of numerical linear algebra libraries is represented by Basic Linear Algebra Subprograms [72] at the end of the 70's. It is considered the Level 1 of this package because of the versions developed after. The Level 1 BLAS provides the implementations of low-level vector operations and a standard interface for them. Typically these operations involve o(n) floating-point operations and o(n) data items moved, where n is the length of the vectors.

The Level 1 BLAS permits efficient implementation on scalar machines, but if we consider the ratio of floating-point operations to data movement, it is too low to achieve an effective use of most vector or parallel hardware. However, the Level 1 BLAS were extensively and successfully exploited by different packages, such as for example LINPACK [41], that enables for the solution of dense and banded linear equation and linear least squares problems, and EISPACK [126], that computes the eigenvalues and eigenvectors of nine classes of matrices.

2.1.1.2 Level 2 BLAS

The Level 1 BLAS does not have a sufficient granularity to enable optimizations provided by high-performance computers that use vector processors. In fact with such machines, there is the need of considering at least matrix-vector operations, while the Level 1 BLAS hides the matrix-vector nature of the operations from the compiler. Therefore, an additional set of BLAS, called the Level 2 BLAS [43], was designed for a small set of matrix-vector operations that occurs frequently in the implementation of the most common algorithms in linear algebra. The Level 2 BLAS involves o(mn)scalar operations, where m and n are the dimensions of the matrix involved. If we consider a vector-processor machine, operations are computed one vector at a time, and data movement are minimized; therefore the memory access are n. Unfortunately, this software arrangement is often not well suited to computers with a hierarchy of memory.

Many of the frequently used algorithms of numerical Linear algebra can be coded as calls to Level 2 BLAS routines; efficiency can then be obtained by using tailored implementations of the Level 2 BLAS routines.

2.1.1.3 Level 3 BLAS

Considering machines having a memory hierarchy, the Level 2 BLAS organization does not make an efficient reuse of data that reside in cache or local memory. For those architectures, it performs better to partition the matrix into blocks and to execute the computation by matrix-matrix operations on the blocks. The Level 3 BLAS [42] is targeted at the matrix-matrix operations required for these purposes. The routines are derived in a fairly obvious manner from some of the Level 2 BLAS, by replacing the vectors involved in the operations with matrices. By organizing executions in this way, it is ensured a full reuse of data while the block is held in the cache or local memory, avoiding excessive data movement. In fact, it is often possible to obtain $o(n^3)$ floating-point operations while creating only $o(n^2)$ data movement.

2.1.2 LAPACK

A step forward has been made with the development of LAPACK, Linear Algebra PACKage [5], that provides routines for solving linear systems, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems. LAPACK routines are written so that as much as possible of the computation is performed by calls to the BLAS.

The original goal of the LAPACK was to integrate two sets of algorithms into a unified systematic library. In fact it makes the widely used EISPACK and LINPACK libraries run efficiently on shared-memory vector and parallel processors. Great efforts were expended to incorporate design methodologies and algorithms, they make the LAPACK codes more appropriate for today's high-performance architectures. In fact LINPACK and EISPACK, based on the Level 1 BLAS, mostly ignore the multi-layered memory hierarchies, and spend a lot of time in data movement instead of in floating-point operations. LAPACK addresses this problem by reorganizing the algorithms to use block matrix operations, such as matrix multiplication, in the innermost loops. Thus LAPACK routines were designed to exploit the Level 3 BLAS, instead of the Level 1 BLAS. Because of the coarse granularity of the Level 3 BLAS operations, their use promotes high efficiency on many high-performance computers.

2.1.3 ScaLAPACK

ScaLAPACK, Scalable LAPACK [16], is a library of high-performance linear algebra routines for distributed memory MIMD computers. It is a continuation of the LAPACK project; in fact it includes a subset of efficiently redesigned LAPACK routines. ScaLA-

PACK represents a great effort in the development and promotion of standards, especially for low-level communication and computation routines.

A characteristic feature of ScaLAPACK is the data distribution applied in the code, i.e. *two-dimensional block cyclic* distribution. The block cyclic data distribution reduces the overhead due to load unbalancing and data movement maximizing the local node performance. It also enables the reuse of the sequential LAPACK library.

The fundamental building blocks of the ScaLAPACK library are distributed memory versions of the BLAS, Parallel BLAS or PBLAS [23], and BLACS [46], Basic Linear Algebra Communication Subprograms, that addresses the communication tasks that frequently arise in parallel linear algebra computations. Also the code organization represents a notable feature of the package. It is really well-known the poster representing the software organization of that library, see Figure 2.1. The components below the line, labeled **Local**, are called on a single processor, with arguments stored on single processors only. The components above the line, labeled **Global**, are synchronous parallel routines, whose arguments include matrices and vectors distributed across multiple processors.

In this way the source code of the top software layer of ScaLAPACK looks very similar to that of LAPACK, this enables a simple exploitation of ScaLAPACK routines.

2.1.4 Self Adaptive Numerical Software (SANS)

Recently, great efforts have been spent in the design and the development of general strategies to enable self-adaptive software. The aim is to optimize software for a specific platform automatically, because the traditional strategy is to produce hand-optimized routines for a given machine. It is a very expensive process, that results unmaintainable in the long-term.

The SANS [40] is a collaborative effort between different projects to optimize software considering the execution environment. The following subsections describe some illustrative examples for linear algebra packages.

2.1.4.1 ATLAS

The ATLAS (Automatically Tuned Linear Algebra Software) project [143] applies empirical techniques in order to obtain portable performance in sequential software. It provides efficient implementations of BLAS and few LAPACK routines. ATLAS exploits code generators, multiple implementations, sophisticated search scripts and robust timing mechanisms in order to optimize the executions for a given architecture.

In this way ATLAS provides machine-specific tuned libraries; considering that a highly tuned code may run multiple orders of magnitude faster than a general coded routine,

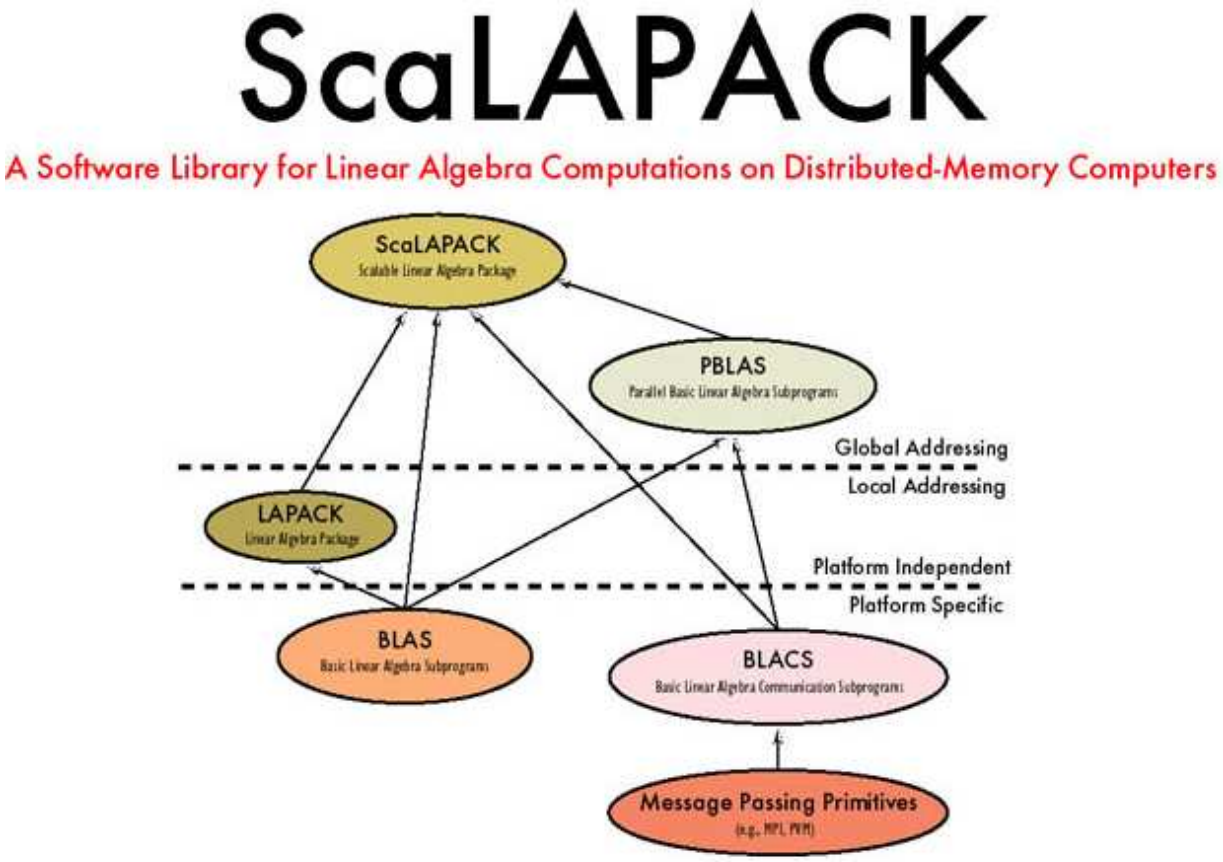


Figure 2.1: Description of ScaLAPACK software hierarchy.

the BLAS implementations are competitive with the machine-specific versions of most known architectures. During installation, ATLAS evaluates architectural characteristics, such as the size of the cache, the number of floating point units and the pipeline length; this leads to an automatic adaptation to a new architecture in a matter of hours. A quite similar approach is applied by the PHiPAC [15], and Fastest Fourier Transform in the West (FFTW) [55] successfully generate highly optimized libraries for sparse linear algebra kernels and fast Fourier transform (FFT).

2.1.4.2 LAPACK for Clusters (LFC)

The LFC project [108] aims to simplify the use of parallel linear algebra software on computational clusters. In fact the LFC software has a serial, single processor user interface, but delivers computing power achievable by running the same problem in parallel on a set of resources of a cluster. The details for parallelizing the user applications are all handled by

the software in a transparent manner. The general goal is to address both computational time and space complexity issues on the user behalf.

Adaptivity is performed by taking into account the resource characteristics of the computational machines and an optimal or near-optimal choice of a subset of resources for the execution of the parallel application is made through scheduling algorithms. LFC also optimizes the parameters of the problem, namely the block size of the matrix. If it is possible to execute the problem faster, LFC maps the problem into a parallel environment, otherwise the application is executed locally optimizing the sequential algorithm.

2.1.5 Considering different environments

In this Section we briefly name few successful projects aimed to use some of the mentioned libraries in distributed and Grid environments, this topic will be considered in further details in Chapters 5 and 6.

The Cactus Project provides a problem solving environment for parallel computation across different architectures [63]. It enables collaborative code development among different groups, and easy access to standard computational capabilities, such as parallel I/O, data distribution, or check-pointing. It includes the PETSc, Portable Extensible Toolkit for Scientific computation scientific library [103], a tool for the numerical solution of partial differential equations. Cactus represents a programming framework developed to enable scientists and engineers to perform the large scale simulations needed for their science.

Also NetSolve, then extended in GridSolve, is a project that aims to bring together disparate computational resources connected by computer networks, [7]. It is a RPC based client/agent/server system that allows remotely access hardware and software components. NetSolve provides numerical routines including dense linear algebra from BLAS, and LAPACK. GridSolve develops a middleware necessary to provide a seamless bridge between the simple, standard programming interfaces, and Grid services.

The VGrADS project, Virtual Grid Application Development Software [136], develops software systems aimed to simplify the development of Grid services and applications, and to ensure high performance and resource exploitation. A first experimentation considers the integration of ScaLAPACK in their system [65], at present the Fault Tolerant Linear Algebra (FT-LA) is included in their applications.

2.2 Peculiar features of numerical model

It is important to summarize the peculiar features in the numerical library model that we considered as key points during the design and the development of PIMA(GE)^2 Lib.

- Software organization and modularity; the identification of building blocks, the separation of their sequential implementation and the identification of the routines that actually introduce the parallelism in the computations, as depicted in Figure 2.1.
- Program readability; the self-documenting quality of code is improved associating widely recognized mnemonic names with mathematical operations.
- Flexibility; users can construct new routines from well-designed parts, as for example LAPACK makes using the BLAS. Thus they are able to add new applications and obtain code extensibility.
- Easiness of use; mathematical operations are provided through a clear interface that enables the actual use of the library. For example the interface of LAPACK and ScaLAPACK looks very similar.

2.3 An example of parallel image processing library: ParHorus overview

The parallel image processing library ParHorus [112], developed by Dr. Frank J. Seinstra, University of Amsterdam, is the parallel version of a sequential library Horus [85], a previous project where the author was involved. ParHorus has been implemented in C and MPI, and supplied with a sequential interface to hide the parallelism of the computation. A "proof-of-concept" partial version of the library is downloadable at:

http://carol.wins.uva.nl/~fjseins/ParHorusCode/ParHorus/

ParHorus is based on the classification provided in Image Algebra [106] for image processing operations. In this way, the author derives a small set of so-called *algorithmic patterns*, which act as templates for a large set of image processing operations with similar behavior. In the specific, ParHorus provides the following set of algorithmic patterns:

- **Unary pixel operation** Operations that elaborate one image applying a unary function, i.e. square root, absolute value, sine;

- **Binary pixel operation** Operations that elaborate two image, i.e. addition, product, etc;
- **Reduce operation** Operations that elaborate one image calculating a collective value, i.e. the maximum, minimum pixel value of an image;
- **Neighborhood operation** Operations that elaborate one image applying a convolution function that involves a neighborhood of each pixel: percentile, median;
- **Generalized convolution** Operations that elaborate one image applying a convolution function that combines together by two binary functions: Gaussian convolution, erosion, dilation;
- **Geometric (domain) operation** Operations that elaborate one image applying a ROI or a domain function, i.e. rotation, translation and scaling;

The concept of algorithmic pattern is crucial for the organization of the parallelism in ParHorus, and it is explained in Section 2.3.2. The ParHorus operations have been implemented in a data parallel fashion, and an automatic parallelization strategy is applied by default in the code. An optimization strategy is applied to guarantee efficient executions of single operation, and sequences of library calls. Thus besides the image processing operations, the author defines a performance model, the Abstract Parallel Image Processing Machine (APIPM), a communication model, and the Lazy parallelization strategy.

2.3.1 Architecture overview

The software architecture consists of six components [113], depicted in Figure 2.2.

Component 1: Parallel image processing library

The core of the ParHorus architecture is a software library of data types and associated operations commonly applied in image processing community. It is largely detailed in the remaining of the Section. Five additional architectural components have been implemented in close connection with the software library itself, in order to implement the parallelization and optimization strategies applied in ParHorus.

Component 2: Performance Models

Each library operation is annotated with a domain specific performance model. The model is based on the definition of an abstract machine, named Abstract Parallel Image Processing Machine (APIPM). The aim of the APIPM is to capture the hardware and software aspects of image processing operations executing on a system. An overview of the APIPM is presented in Section 2.3.5.

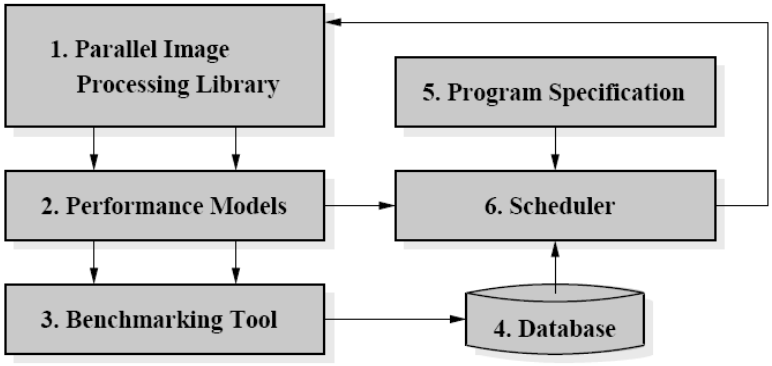


Figure 2.2: *ParHorus software architecture overview.*

Component 3: Benchmarking tool

A set of benchmarking operations is used to obtain performance values for the model parameters. The combination of the high-level APIPM-based performance models and the benchmarking routines allows the intra-operation optimization automatically. This is performed by the scheduling component, described below.

Component 4: Database of benchmarking results

All benchmarking results are stored in a database of performance values.

Component 5: Program specification

It is assumed that an algorithm specification is provided in addition to the program itself. Such specification closely concerns a concatenation of library calls, to enable the inter-operation optimization.

Component 6: Scheduler

Taking into account performance models, benchmarking results, and the algorithm specification, the scheduler applies the optimization and parallelization policies. It is obtained through the definition of a finite state machine, which allows the application of a straightforward and cheap run time method (called *Lazy parallelization* described in Section 2.3.7) for communication cost minimization.

2.3.2 Parallelism in ParHorus

As already said, ParHorus is the parallel version of the sequential Horus library. Additional functionalities have been implemented in order to reuse as much as possible the existing sequential code. We consider the first component of the previous Section; it is the core of the software architecture, and in turn it consists of four logical components, see Figure 2.3.

Component C1: Sequential Algorithmic Patterns

The component C1 consists of the sequential algorithmic patterns introduced at the beginning of the Section. In this component, the algorithmic patterns are implemented as a sequence of sequential routines. Apart from possible memory operations, such sequence represents a *parallelizable pattern*, described in Section 2.3.3, which incorporates only the instructions that can be applied in the sequential algorithmic pattern and in the implementation of the corresponding parallel algorithmic pattern as well.

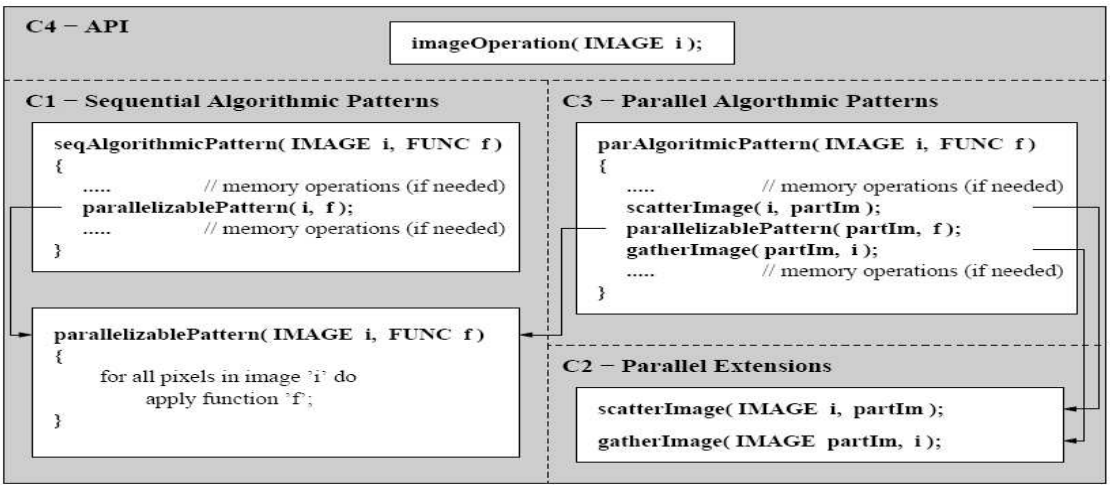


Figure 2.3: *ParHorus code overview.*

Component C2: Parallel Extensions

The component C2 effectively introduces the parallelism in the library through MPI. It addresses different parallel aspects, as the partition of data structures, and the exchange of data among processes, and includes:

1. routines for data structure *partitioning*, used to indicate which data parts should be processed by each node;

2. routines for data *distribution* and *redistribution*, used for the actual distribution and collection of data structures;

3. routines for *overlap communication*, used to exchange the so-called ghost regions among neighboring nodes.

During the computation, the parallel processes are organized in a logical grid of up to 3 dimensions. Functions belonging to this component are totally hidden to the users through a sequential API, see component C4.

Component C3: Parallel Algorithmic patterns

The implementation of each parallel algorithmic pattern is obtained combining communication operations from component C2 and the algorithmic pattern of component C1. Communications among processes involves non-local data (i.e. data residing on other processes) necessary for the execution of the parallelizable pattern.

Component C4: Fully sequential API

The same Application Programming Interface (API) of the original sequential Horus library is used to provide the ParHorus functionalities. As a consequence, the API does not contain references to the parallel processing capabilities, and users are not involved in the parallelism of the computation.

2.3.3 Parallelizable pattern

As already said, starting from the abstractions of Image Algebra [106], it is possible to recognize a set of *operation classes* commonly used in image processing. They provide the so-called *algorithmic patterns*, and they act as template for the image processing operations with similar behavior.

For each algorithmic pattern, the author defines a so-called *parallelizable pattern*. It represents the maximum amount of code of an algorithmic pattern that can be performed both sequentially and in parallel. In other words, each pattern represents the maximum amount of work that can be performed without the need for communication, therefore, all internal data accesses must refer to data local to the processes.

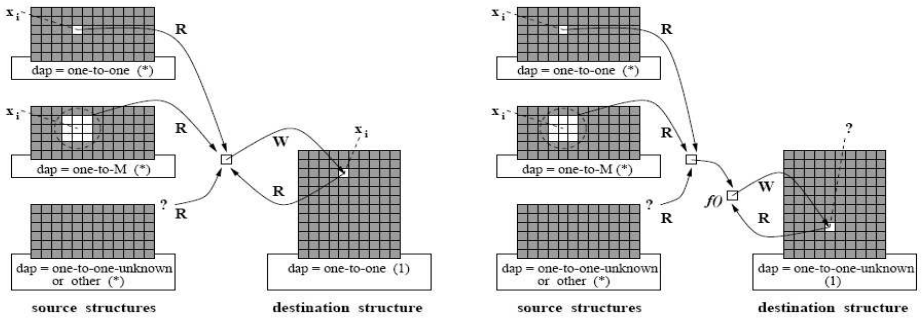
In order to give a general description, a parallelizable pattern is defined as "a sequential generic operation that takes zero or more source structures as input, and produces one destination structure as output. Each pattern consists of n independent tasks, where a task specifies what data in any of the structures involved in the operation must be acquired (read), in order to update (write) the value of a *single* data point in the destination structure. In each task read access to the source structures is unrestricted, as long as no

accesses are performed outside any of the structures' domains. In contrast, read access to the destination structure is limited to the single data point to be updated". All n tasks are tied to a different task location x_i con $i \in \{1, ..., n\}$, [116].

Depending on the number of pixel accessed in the acquisition phase of each task, it is possible to intuitively understand the definition of four *data access pattern types*:

- *One-to-one*: For a given data structure, in each task no data point is accessed other than x_i.
- *One-to-one-unknown*: For a given data structure, in each task not more than one data point is accessed. In general, it is not equal to x_i.
- *One-to-M*: For a given data structure, in each task no data points are accessed out of the neighborhood of x_i.
- *Other*: For a given structure, in each task either all elements are accessed, or the accesses are irregular or unknown.

A parallelizable pattern requires the specification of the access pattern type for each data structure involved in the operation. The destination structure is accessed to update a single value. An example to better understand the purpose of the different access pattern type is given in Figure 2.4.



(a) Parallelizable pattern, type 1.

(b) Parallelizable pattern, type 2.

Figure 2.4: *Two parallelizable pattern type.*

They differ in the type of combination operation that is allowed. In a parallelizable pattern of type 1, no restrictions are imposed on the combination operation. In a type 2 pattern, the final combination must be performed by a function $f()$ -that is associative and commutative-.

They give a generalization of a large set of sequential image processing routines that are the commonly applied in image processing research, but they do not describe all the possible operations. For the algorithmic patterns that can be organized according to these parallelizable pattern types, the author defines a standard parallelization strategy for any algorithmic pattern that corresponds to one of the two parallelizable pattern types.

2.3.4 The parallelization strategy

The starting point is to define the parallel implementation of each algorithmic pattern. An operation is implemented by instantiating the corresponding generic algorithmic pattern with proper parameters, including the operation to apply. The parallel implementation of each algorithmic pattern depends on the corresponding parallelizable pattern types.

For example to execute the parallelizable pattern type 1, each process does not consider the neighborhood of a data, therefore it has to be provided with a *non-overlapping* partial destination structure. Instead, to execute the parallelizable pattern type 2, each process has to locally create a *fully overlapping* destination structure. Similar considerations are the base for the implementation of each algorithmic pattern.

In order to intuitively understand how actually the parallelization strategy performs, we provide two examples of algorithmic patterns and their parallel implementations: a reduce operation and a generalized convolution.

2.3.4.1 Parallel reduction

A sequential generic reduction operation produces a scalar or vector value. Depending on the kind of reduction, this value will represent for example the maximum or the minimum pixel of the image, the sum or the product of all the pixel of the image.

The reduce operations can be parallelized considering the access pattern type "one-to-one" for the input image, and the "one-to-one-unknown" for the single result value. The parallel implementation of a generic reduction operation follows directly from these considerations. The image can be subdivided among the parallel processes, with non-overlapping partial destination structure, the execution with 2 processes is depicted in Figure 2.5.

2.3.4.2 Parallel generalized convolution

A generalized convolution combines an input image with a given kernel, the kernel and the combining function characterize the specific convolution to apply. The value of each pixel $x(i)$ in the output image depends on the neighborhood of the pixel $x(i)$ in the input image,

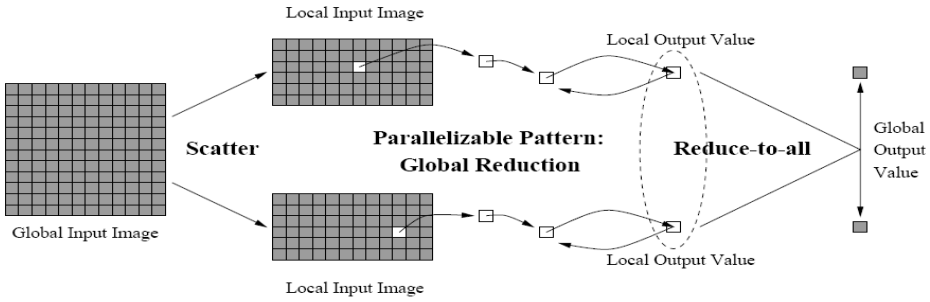


Figure 2.5: *Example of reduce operation executed considering two processes.*

and the values of the kernel. This situation is presented in Figure 2.6, and it underlines the necessity to consider the "border problem", i.e. to create a neighborhood for the pixel on the border of the image.

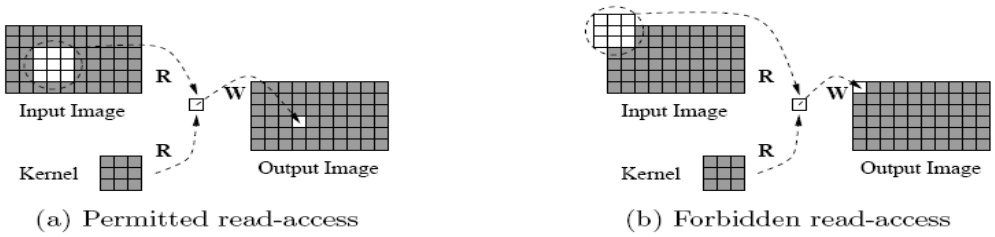


Figure 2.6: *Example of a (simplified) generalized convolution executed considering two processes.*

ParHorus considers the so-called *scratch border* around the original input image, and applies predefined border handling strategy (e.g., mirroring or tiling) to fill the border; it is specified by the user. As each local scratch image has a one-to-M access pattern, an overlapping scatter of the global input image is required. The parallelizable pattern is executed, producing local result images that are gathered to obtain the global output image. The Figure 2.7 gives a simplified view, as some steps of the operation are not shown.

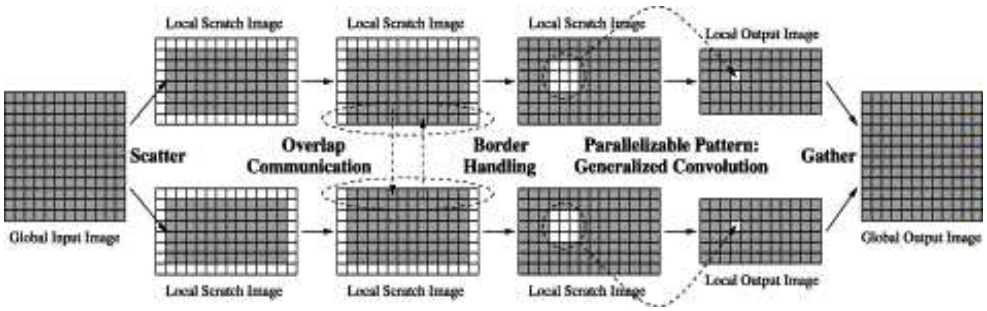


Figure 2.7: *Example of a (simplified) generalized convolution executed considering two processes.*

2.3.5 Abstract Parallel Image Processing Machine

To increase the efficiency of the applications developed with ParHorus, a performance model for run-time cost estimation has been considered. Typically performance models represent a key topic in the software development, and their accuracy allows optimization decisions. However, estimation accuracy implies a great complexity that may degrade the application performance.

ParHorus applies *semi-empirical modeling* technique based on abstraction, simple modeling, and domain-specific measurement. It is composed by three essential elements:

1. A high level abstract machine for parallel low level image processing (APIPM), and a related set of instructions.
2. A linear performance model related to each user-callable library operation.
3. A benchmarking method to capture essential cost factors not explicit in the model.

In APIPM the common hardware characteristics are based on the definition of *abstract hardware components*. In addition, the behavior of routines is obtained with a related *instruction set*. The APIPM instructions are parameterized, since the execution of an instruction is often dependent on the values of its operands, [116].

An APIPM consists of one or more identical abstract sequential image processing machines (ASIPMs), each consisting of four related components: (1) a *sequential image processing unit* (SIPU), capable of executing APIPM instructions, one at a time, (2) a *memory unit*, capable of storing (image) data, (3) an *I/O unit*, to move data among the memory unit and external sensing or storage devices, and (4) *data channels*, to transfer data between ASIPM units and external devices. The description of the APIPM reflects homogeneous distributed memory MIMD machines, but it is designed for image processing tasks.

The APIPM instruction set consists of four classes of operations: (1) *generic image instructions*, i.e. the specialized parallelizable patterns of Section 2.3.3, (2) *memory instructions*, for allocation and copying of (image) data, (3) *I/O instructions*, to move data between memory and external devices, and (4) *communication instructions*, for exchanging data among ASIPM units.

ParHorus operations are seen as concatenation of APIPM instructions, and their performance are obtained considering the execution time of APIPM instructions. Similarly, the execution time of an application is given by the execution time of each library operation involved in the application.

The semi-empirical modeling approach requires an additional benchmarking phase. Benchmarking results need to be obtained only once on a specific platform, since a database of benchmarking results is provided in the software architecture, as we described in Section 2.3.1. Therefore if the platform is not upgraded, the same set of measurement results can be used for estimation of any application.

2.3.6 Communication model

Since the semi-empirical model does not take into account the communication operations, the author defines an extended model for point-to-point communication. In fact, even if MPI provides collective optimized communications, in ParHorus communication operations have been implemented considering point-to-point communications.

The communication model is called P-3PC (or *the parameterized model based on the three paths of communication*), [114]. It is close to other models present in literature (e.g., the Postal Model, LogP, and LogGP), it considers communication patterns used in data parallel image processing applications. It predicts the communication costs of any domain decomposition.

The author adds three elements to such models: 1) P-3PC recognizes the difference in the time to send or to receive a message, and the complete end-to-end delivery time. 2) P-3PC makes a distinction between communicating data stored either contiguously or non contiguously in memory. 3) P-3PC does not assume a strictly linear relationship between the size of a message being transmitted and the communication costs.

The model introduces the notion of the three paths of communication, and assumes that the cost of transferring a message from a sender to a receiver is captured in three independent values:

1. Tsend: the cost related to the communication path at the sender (i.e., the time required for executing the SEND instruction).

Import(A); UnPixOp(A,B); BinPixOp(A,B,C); Export(C); Delete(A); Delete(B); Delete(C);	Import(A); Scatter(A,locA); UnPixOp(locA,locB); Gather(locB, B); Delete(locA); Delete(locB); Scatter(A, locA); Scatter(B, locB); BinPixOp(locA,locB,locC); Gather(locC, C); Delete(locA); Delete(locB); Delete(locC); Export(C); Delete(A); Delete(B); Delete(C);

Table 2.1: *An abstract sequential application, and the equivalent program considering a default parallelization.*

2. Trecv: the cost related to the communication path at the receiver (i.e., the time required for executing the RECV instruction).

3. Tfull: the cost related to the full communication path. This value represents the end-to-end delivery time, or the time from the moment the sender initiates a transmission until the moment the receiver has safely stored all data and is ready to continue.

For each path, the author defines a model that is "parametrized" in order to describe also the memory layout in the communicating processes.

2.3.7 Lazy parallelization

The *Lazy parallelization* is the strategy applied in ParHorus to obtain a global performance optimization [115]. Its aim is to automatically parallelize applications by inserting communication operations only when they are necessary. The lazy parallelization is based on the definition of a *finite state machine*, (fsm). The fsm is composed by two essential points:

Import(A); UnPixOp(A,B); BinPixOp(A,B,C); Export(C); Delete(A); Delete(B); Delete(C);	Import(A); Scatter(A,locA); UnPixOp(locA,locB); BinPixOp(locA,locB,locC); Gather(locC, C); Delete(locA); Delete(locB); Delete(locC); Export(C); Delete(A); Delete(B); Delete(C);

Table 2.2: *An abstract sequential application, and the equivalent program considering the lazy parallelization.*

1. a set of states, corresponding to a valid internal representation of a distributed data structure at run time,
2. a set of state transition functions, defining when a valid data structure representation is transformed into another valid representation.

The lazy parallelization works well in many situations, however it does not guarantee to produce the fastest version of a program, and does not take into account the APIPM performance models. To overcome these problems, the author proposes an extended technique, which requires an *application state transition graph* (ASTG) for the program under consideration.

The strategy is based on the idea of create an optimized default parallelization. For each library operation, a default parallelization strategy has been implemented. The default algorithm expansion is guaranteed to produce a legal and correct parallel version of any sequential program developed by the users. It is because the function calls in the sequential application are replaced by an equivalent sequence of one or more (parallel) operations. However, the resulting program is not guaranteed to be time-optimal, it is likely that redundant communication phases will be performed. An example of such situation is reported in Table 2.1

To avoid this situation, the author starts from the idea that each communication or memory management operation of a default parallelization is redundant, unless the fsm proves

otherwise. It happens when, a default communication or memory management operation leads to a data structure state inconsistency. Therefore, the idea is to apply a default parallelization strategy, then remove all the communication and memory management operations, verify if these removals have introduced data structure state inconsistency. In this case re-insert the operation that causes the fault. The Lazy parallelization behavior is reported in Table 2.2.

2.4 Peculiar features in image processing

The ParHorus library represents an interesting example in image processing community. In this Section, we summarize the features that we considered as key points in the design and the development of PIMA(GE)^2 Lib.

- the classification of the image processing operations that could be interesting for research community. Indeed, the identification of the most common low level operations to produce meaningful results;
- the definition of the algorithmic pattern to group together the operations with a similar behavior; in this way it is possible to avoid code redundancy;
- the definition of the corresponding parallelizable patterns, that enable an easy parallelization of the user's applications.

2.5 Peculiar features in PIMA(GE)^2 Lib

During the design and the development of PIMA(GE)^2 Lib, we try to merge the good features of the previous examples. Therefore the characteristics outlined during this Chapter have been taken into account in the design of PIMA(GE)^2 Lib and in the definition of the strategy to apply:

- the organization and modularity of the code, as for example in ScaLAPACK, in this way it is possible to derive many important features, such as portability, extensibility, and efficiency;
- easiness of use, i.e to provide the operations through a simple interface and considering intuitive widely recognized mnemonic names; in this way inexpert users can utilize library functions;

- transparency of parallelization and optimization policies, this point is really close to the previous one, and the definition of an effective interface is exploited to achieve this goal;
- the classification of the image processing operations and the definition of the algorithmic patterns performed in ParHorus. In this way, we are able to define a hierarchy of image operation classes, and to implement an optimization policy for the single class and the applications developed by the users.

These topics will be detailed in the next Chapter, which is completely dedicated to the PIMA(GE)^2 Lib description. However, in this Section we outline further topics we investigated during the development of the library:

- an efficient implementation of PIMA(GE)^2 Lib I/O operations. Typically libraries do not consider the exploitation of advanced parallel tools in the I/O phase; starting from these remarks, we performed a study about I/O aspects in the image processing, where this topic received only a little attention.
- an efficient execution of PIMA(GE)^2 Lib considering heterogeneous environments. In fact, the parallelization policy applied in ParHorus works well on homogeneous cluster, but suffers from the different computational power in an heterogeneous cluster. In PIMA(GE)^2 Lib this topic has been taken into account to distribute the work, and to avoid load unbalancing and a long waiting time.

Chapter 3

A parallel image processing library for local cluster

In this Chapter we describe PIMA(GE)^2 Lib, the Parallel IMAGE processing GEnoa Library. Starting from the consideration of the previous Chapter, we analyze the main features of the library, giving a detailed description of each single point. We also present some of the experimentations performed during implementation of PIMA(GE)^2 Lib, we deeply discuss these points as well.

3.1 PIMA(GE)^2 Lib general design

PIMA(GE)^2 Lib provides robust and efficient implementations of the most common image processing operations, according to the classification provided in Image Algebra [106]. In particular, we considered the operations classified in literature as *low-level image processing*. During the implementation of the library, we started from two simple considerations:

- a large classes of image processing operations requires a given and spatially localized portion of the input image in order to compute the required output image. In the simplest case, an output image is computed simply by processing single pixels of the input image or, more in general, only neighborhood pixels are used to compute an output pixel. For this reason it is possible to compute the output data largely in independent way and therefore in parallel;
- a large number of image processing routines uses a relatively small number of *computational kernel*, where each kernel uses a similar methodology to implement computational engines. Therefore, it is possible to consider each image processing operation as belonging to a group, and define image operation classes.

It means that most of the image processing functions are easily parallelizable in a data parallel fashion. Grouping together the operations, we can outline their computational behavior and ensure few schemes, acting as guide in the parallelization strategy. The parallelization policy applied in PIMA(GE)^2 Lib is described in Section 3.4, we also consider an optimized exploitation of heterogeneous parallel resources.

PIMA(GE)^2 Lib has been implemented using C and MPI-CH 2 [89], the library allows the development of compute intensive applications, achieving good speed up values and scalability, as we present in Section 3.5. We considered the exploitation of advanced parallel tools to improve the library performance, their main contribution is obtained in the data management, as described in Sections 3.4.3.2, and 3.4.3.5.

The library elaborates bidimensional and tridimensional data sets, and may perform operations both in sequential and in parallel, depending on the user requirements. In the second case, the management of the parallelism in PIMA(GE)^2 Lib is obtained with a set of functionalities, which are transparent to the users. The library code encapsulates parallel execution behind a sequential interface. In this way PIMA(GE)^2 Lib shields users from the intrinsic complexity of a parallel application, and becomes an effective tool to employ in the development of complex image processing applications. The definition of the sequential Application Programming Interface (API) of the library is described in Section 3.3.

The structure of PIMA(GE)^2 Lib code is similar to the one adopted by the parallel libraries for linear algebra described in the previous Chapter. In fact, during the development of the library, we consider a clear distinction between sequential code and code introducing the parallelism. This separation leads to the same situation, depicted in Figure 2.1.

3.2 Image Processing Operations

The operations provided by PIMA(GE)^2 Lib are classified in literature as low-level operations. They represent the operations commonly used in image processing, and require additional structures to enable the effective executions. For this reason, they imply the definitions of abstract data types and of the corresponding primitives to manipulate them.

During the classification of the library operations, we introduce conceptual classes in order to group together functions according to different rules. To group together image processing operations represents a well-established code arrangement, already presented in literature ParHorus [113], PIPT [129], VSIPL ++ [73], EASY-PIPE [93] and Oliveira et al. [96]. We organize the classes in logical levels, obtaining in this way a hierarchy of image processing operation classes; levels are described in the remaining of the Section.

This classification does not put constraints on the actual code implementation, because it is a conceptual step aimed to obtain an object oriented structure of PIMA(GE)^2 Lib.

Indeed it represents a peculiar feature of the library, and leads to avoid code redundancy and to ensure the maintainability of the code. Moreover, this code organization supports the achievement of several goals in the Thesis:

- The definitions of the parallelization policy and the user-friendly interface of the library. Indeed it is simpler to manage few and well-defined classes, rather than consider dozens of single functions. Also from the user's point of view, this results in a code that is simple to use. The outlined hierarchy provides a model for the codification of an adequate interface also in the integration in distributed environments;

- The extensibility of the operations provided by PIMA(GE)^2 Lib. In fact, it is possible to add further operations included in the image processing operation classes, or to define new classes of operations. For example, we develop a set of high level functions properly combining and customizing some original library operations for TMA analysis. They represent a specialization of PIMA(GE)^2 Lib for the analysis of TMA images, and they are described in Section 3.5;

- The interoperability and the reusability of the library code in heterogeneous and distributed environments; we actually exploit the hierarchical structure of the code also to enable the reuse of the image processing operations in distributed environment. For example, the whole library has been integrated in a server developed using CORBA, as described in Chapter 5; and the specialized set of operations to elaborate TMA images has been integrated in Grid portal, as described in Chapter 6.

- The integration of the library code with local services and distributed protocols provided by specific middlewares. Actually we enable the exploitation in a transparent manner of advanced parallel tools for data acquisition; thus it is possible to acquire data considering a local I/O or a parallel I/O on architectures as clusters, see Section 3.4.3.2. Considering a Grid environment, we also enable the exploitation of specialized Grid I/O protocols; this aspect is described in Chapter 6;

- The integration and the tuning of the parallelization policy applied in the library. We already consider an optimized version of the parallelization policy in heterogeneous local cluster; the strategy is aimed to ensure load balancing even in such architectures, see Section 5. Moving toward distributed and Grid environments, we employ this strategy to achieve the same goal. In particular on the Grid, we consider its use for the scheduling of the applications on the available resources, also this aspect is discussed in Chapter 6.

3.2.1 The basic image objects

The basic elements of the hierarchy are represented by the fundamental data structures of the library:

- IMAGE, used to store images;
- KERNEL, used to store convolution kernels;
- MATRIX, used to store geometric matrices.

At an abstract level, we may assimilate them to data types. An object instance correspond to program variables. Each object is associated with a set of suitable function for input, output, data distribution and collection. These functions may be called "basic functions" or "basic methods" of the library. They are summarized in Table 3.1.

IMAGE	**MATRIX**	**KERNEL**
createImageOnOne	createMatrix	createGaussianKernel1d
deleteImageOnOne	deleteMatrix	deleteGaussianKernel1d
createImage	printMatrix	createGaussianKernel2d
deleteImage	translate2d	deleteGaussianKernel2d
copyImage	scale2d	createKernel
readImageFromFile	rotate2d	createKernelFromData1D
writeImageToFile	copy	deleteKernel
	transp	
	inv	

Table 3.1: *Objects belonging to the first layer of the* PIMA(GE)^2 *Lib and "basic functions" implemented to enable their proper management and manipulation.*

A part of this software layer is hidden to the users. In particular, users are allowed to manage IMAGEs, and to define specific KERNELs to obtain a specific convolution or other algorithms, for example a Sobel filter; but they are not involved in the manipulations of MATRIXs, since they are embedded in the Geometric operations.

3.2.2 The image operation classes

The image operation classes are the low level operations that act upon the basic image objects. Classes are identified using the schematic organization made in Image Algebra [106]. Operation of the same class have the same "sort", i.e. acts on the same data types and produces the same output. Image operation classes are six:

1. **Point Operations** Operations that take in input only one IMAGE object, and apply an unary operation, such as square root, absolute value, threshold;

2. **Reduction Operations** Operations that take in input only one IMAGE object, and a collective value, i.e. the maximum, minimum pixel value;

3. **Image Arithmetic** Operations that take in input two IMAGE objects and apply a binary operation, i.e. addition, product, etc;

4. **Geometric Operations** Operations that take in input one IMAGE object and one MATRIX object, and apply a Region Of Interest (ROI) operation as restriction, or a domain transformation as a scaling;

5. **Convolution Operations** Operations that take in input one IMAGE object and one KERNEL object, and apply a convolution, as Gaussian convolution, erosion, dilation.

6. **Differential Operations** Operations that take in input one IMAGE and perform differential operators, i.e. Hessian, gradient, Laplacian. These operations are not actually low level operations.

Image operation classes are related to algorithmic and parallelizable patterns.

3.2.3 The image processing library

An image processing library is a top level entity that wraps together Image processing objects, functions, and Image operation classes. In same sense we may think to a library like to a container, similar properties are portability, thread-and process- safety, re-entrancy [142]. We define two fundamental methods on Image Processing Library, open and close:

- **OpenLib**, it is a single method that is responsible for the initialization of the library functionalities, in particular it is aimed to the start-up of the parallel environment, and the information related with the parallel processes involved in the computation;

- **CloseLib**, it is a single function specular to the *OpenLib*, in fact it is responsible for the termination of the library operations, and of all parallel processing as well. Any library calls made after this function raise an error.

Another fundamental method of this level is a conceptual element, that enables to include the classes of the previous level:

- **Operations**, it contains the image processing operations provided by the library and groups together the objects belonging to the previous level,

In this way image processing library embeds the image operation classes.

3.2.4 Wrapping all together in a hierarchy

Object, Operations, and Library are strictly interconnected. It is useful to think to them as entities and describe their relationships in a hierarchical way as depicted in Figure 3.1. The

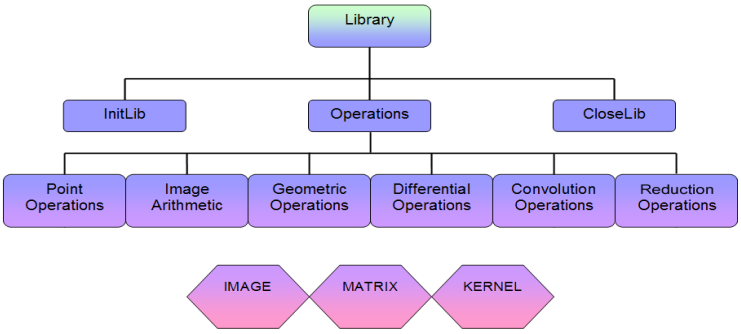


Figure 3.1: *Overview of the* PIMA(GE)^2 *Lib hierarchy of image processing operations. The hidden code of the library is not included in the classification, and thus it is not depicted.*

hidden code level of PIMA(GE)^2 Lib is not included in the library code classification since it is not strictly related with image processing. In fact the library hidden code concerns the management of secondary data structures, as described in this Section, and with the management of the parallel environment and the optimization policy.

3.3 A user friendly API

To allow a simple and coherent use of PIMA(GE)^2 Lib, the most relevant element is the definition of effective and flexible interface, that allows the development of efficient image processing applications. It can be achieved through a natural codification of the library hierarchy, which directly provides a model to derive interfaces with the aforementioned properties. Since the hierarchy definition is completed, the PIMA(GE)^2 Lib interface implementing phase is totally planned and easily obtained. Furthermore, the hierarchy obtained through the classification, provides a model also for the definition of the tools we

develop in distributed and Grid environments. In box code 3.1 we provide few examples of PIMA(GE)^2 Lib operation APIs.

Listing 3.1: *API of the main operations of* PIMA(GE)^2 *Lib.*

```
1 //Functionalities mandatory to enable the use of the library

3   ERROR OpenLib(int *pArgc, char ***pArgv);
    ERROR CloseLib();
5
  //Functionalities to IMAGE management
7   IMAGE * ReadImageFromFile(char *pFileName, int pWidth,
  int pHeight, int pDepth);
9   ERROR WriteImageToFile(char *pFileName, IMAGE *pImPtr);

11 //Functionalities to KERNEL management
    KERNEL * CreateKernel(int kerWidth, int kerHeight, int kerDepth);
13   void   DeleteKernel(KERNEL *pKerPtr);

15 //Point Operations (with a constant parameter)
    ERROR UnaryPixOpC(IMAGE *pImgPtr,  PIXEL pVal, UNARY_OP_C_ID pOpId);
17   ERROR ReductionOp(IMAGE *pImgPtr, REDUCTION_OP_ID pOpId, ARITH *pRetVal);

19 //Image Arithmetic (with an IMAGE parameter)
    ERROR BinaryPixOpI(IMAGE *pImgPtr, IMAGE *pArgImgPtr, BINARY_OP_I_ID pOpId);
21
  //Convolution operations
23   ERROR ConvolutionOp(IMAGE *pImgPtr, KERNEL *pKerPtr, CONV_OP_ID pOpId);

25 //Geometric operations (ROI)
    ERROR GeoRoiOp(IMAGE *pImgPtr, IMG_COORD pBegin, IMG_AXES pNewDim,
27 PIXEL *pBackground, GEO_ROI_OP_ID pOpId, IMAGE **pResPtr);
```

The API reflects the algorithmic pattern of the library. From the user's point of view, it allows an easier management of the library operations, since he is not longer involved with a large number of functions, but he has to consider and to handle a small set of clear and well-defined classes. The single operation may be implemented through the respective algorithmic pattern with the proper parameters, that has to include the single operation in itself. For example, considering the Reduction operations:

Listing 3.2: *An example of library operation.*

```
1  ReductionOp(IMAGE *pImgPtr, REDUCTION_OP_ID pOpId, ARITH *pRetVal);

3 // Where REDUCTION_OP_ID is defined as

5 typedef enum {
        OP_SUM_R,
7       OP_PROD_R,
        OP_MIN_R,
9       OP_MAX_R
  } REDUCTION_OP_ID;
11 %\end{verbatim}
```

Through the parameter `REDUCTION_OP_ID pOpId`, it is possible to specify the kind of reduction to apply, i.e. calculate the minimum or maximum pixel value of an image, calculate the sum or the product of the pixels in an image. In the C programming language, a function can be specified by providing a pointer to it. Unfortunately, in a distributed memory computing environment, a function pointer is not a meaningful way to specify functions on remote compute nodes. Therefore considering the future directions of the Thesis, we avoided this choice, and we considered an *enumerative* data type to specify the possible operations for each algorithmic pattern.

The API of each operation appears completely sequential. The parallel environment, MPI processes, and the data partition or communication never appear in the API of the functions. Thus, users could not be aware of the parallelism of the computation, neither are involved in the difficulties of parallel programming. However, they achieve high performance executions, simply developing sequential C code. The API actually calls proper functions, and hides a layer of communication and transport data across processes. In practice, the sequential API of the library hides an appropriate sequence of function calls, which depends on the different algorithmic patterns.

3.4 Adding an optimized parallelization policy

In order to define a proper parallelization policy for the image processing operations, we started from several remarks already mentioned. As described in Section 3.1, most of image processing operations are easily parallelizable in a data parallel fashion. Thus, an evident key issue is to identify a proper distribution of the global data; this topic is deeply analyzed in Section 3.4.3.5. As presented in Section 3.2, we selected the image processing operations, and we group them together in image operation classes. We also define a sequential API for the library, see Section 3.3, to hide the parallelism of the computation and ensure the easiness of use of PIMA(GE)^2 Lib.

Import(A); UnPixOp(A,B); BinPixOp(A,B,C); Export(C); Delete(A); Delete(B); Delete(C);	Import(A); Scatter(A,locA); UnPixOp(locA,locB); Gather(locB, B); Delete(locA); Delete(locB); Scatter(A, locA); Scatter(B, locB); BinPixOp(locA,locB,locC); Gather(locC, C); Delete(locA); Delete(locB); Delete(locC); Export(C); Delete(A); Delete(B); Delete(C);

Table 3.2: *A sequential application, and the equivalent program considering a default parallelization. Almost all the communication and memory operations are unnecessary and redundant.*

In this kind of situation, i.e. the parallelism is hidden and completely managed by the library, the definition of a robust parallelization policy is necessary. It is likely to parallelize the code implementing a default parallelization, it means that each operation performed is considered in an atomic way, and is executed in parallel but in *isolation*. Therefore, for each operation, data are distributed among the parallel processes, elaborated in parallel, and collected at the end of the operation. It represents a robust parallelization policy, but it is inefficient if the operations are applied to the same data. An example of such situation is reported in Table 3.2. In PIMA(GE)^2 Lib, we want to enable the efficient concatenation of the operation applied to the same image.

The first step is the definition of a parallel implementation for each operation, after that, it is necessary to define further smart rules aimed to improve the execution of a sequence of operations. For this reason we considered three main topics, which characterize the optimized parallelization policy applied transparently in PIMA(GE)^2 Lib; they are:

- the definition of an efficient parallelization policy of the single operations considering load balancing. Adopting the ScaLAPACK terminology, it may be seen as the definition of optimized Building Block for the library and their efficient execution;
- the definition of the efficient concatenation of such Building Blocks, where we avoid

useless data movement among parallel processes, and an efficient exploitation of the parallel environment;

- the employment of effective tools to speed up the data acquisition and distribution phases, since a great impact on the application efficiency is given by a proper data management.

Further experimentations have been considered to define other important features to improve the efficiency of the overall code. Among the topics we discussed: the header of the functions, the main data structures of the library, the arrangement of the cyclic computations, the cache memory exploitation. However, only few efforts were spent in these directions, and we can not mention them as a part of the optimization policy. In the reminder of the Section, we discuss the single point of the previous list, while an example of use and the speed up values achievable using the library are presented in Section 3.5.

3.4.1 Building Block optimization

As already said, the first step is the definition of a parallel implementation for each operation. Considering the image operation classes, we obtain a restrict number of computational schemes, which act as a guide for the definition of the parallel implementation. Actually, the operations belonging to the same class are parallelizable in the same way and they adopt the same parallel behavior. This consideration leads to define an effective parallel execution for each class.

In order to achieve this goal, we adopt the definition and the parallelization made in the ParHorus library, [113], deeply described in Section 2.3.3. Starting from these definitions, we make several modifications to the policy; they mainly concern data distribution and load balancing.

With respect to data distribution, we prefer to assume the same I/O pattern for each image operation class. In this way we avoid further communications during the processing, but we do not apply the optimal parallelization policy to all image operation classes.

Considering load balancing we introduce a further policy to ensure effective execution on heterogeneous cluster. In fact, the parallelization policy applied in ParHorus works well on homogeneous cluster, but suffers from the different computational power in an heterogeneous cluster. In PIMA(GE)^2 Lib this topic has been taken into account to distribute the work, and to avoid load imbalance and a long waiting time. The strategy is detailed in the next Section.

3.4.1.1 Enabling load balancing on heterogeneous resources

As already said, the parallelization policy of the Building Blocks works well on homogeneous clusters, but it performs poorly considering heterogeneous clusters. Thus we define a further policy to exploit the computational power of the single heterogeneous node and to actually ensure load balancing.

Considering an heterogeneous cluster, the first step is to classify the computational nodes on the base of their relative computational power. Typically a reference node is fixed, and the other nodes are sorted with respect to such node. The relative velocity of the single node is determined using a data set and a benchmark function. It is supposed that

$$V_{ref} = 1 = \text{relative velocity of reference computational node}$$

The reference node could be external to the available nodes of the cluster; for each i node of the cluster, its relative velocity is:

$$V_i = S_i * V_{ref}, \qquad S_i > 0$$

and such that for the considered data set and benchmark function,

$$V_i = \frac{T_{ref}}{T_i} = S_i$$

where

$$\begin{array}{lll} S_i = 1 & \text{if} & V_i = V_{ref} \\ S_i > 1 & \text{if} & V_i > V_{ref} \\ S_i < 1 & \text{if} & V_i < V_{ref} \end{array}$$

where T_i is the time spent by the i node to execute the benchmark function on the benchmark data set, and T_{ref} is the time spent by the reference node to execute the benchmark function on the benchmark data set.

Considering a data parallelism, the work load of each node $load_i = l_i$ is proportional to the amount of data that it has to elaborate

$$size(data_i) = D_i \qquad (l_i \approx D_i)$$

On the other hand, the efficiency of a parallel computation is given by a proper load balancing for each node. In an homogeneous cluster, the load balancing is obtained if

$$D_i = D_j \qquad \forall i, j \text{ computational nodes}$$

In an heterogeneous cluster it is intuitive to think, easy to demonstrate experimentally, and largely asserted in literature [71], that a parallel algorithm is efficient when the work load of each node is proportional to its relative velocity, i.e.

$$S_i \approx D_i$$

or

$$\frac{S_i}{S_j} \approx \frac{D_i}{D_j} \qquad \forall i, j \text{ computational nodes}$$

Different strategies have been proposed to outline the optimal solution, or heuristics to bring near optimal solutions; they are aimed to obtain the suitable data partition on heterogeneous architecture for linear algebra problems [18]. However, to introduce such strategies could result difficult and heavy for mainly two reasons: an algorithmic aspect, i.e. the computational cost of the heuristics for the partition of the data set, and a software engineering aspect, i.e. the cost of the modifications to the library code. Actually to enable a suitable data partition, a parametric partition has to be introduced, it is automatically tuned by the heuristics; in particular it is necessary to evaluate the impact of such modifications on the functions aimed to the distribution and communication of data.

An alternative strategy to this kind of heuristics is the introduction of the *farm-on demand paradigm*, applied considering a fine coursed of the tasks. This strategy leads to achieve good or even optimal results, [25], [11]. However, also in this case the introduction of the strategy could require a consistent modification to the code.

A further strategy is based on an homogeneous data partition considering a number of processes higher than the actual number of computational node; the tuning of the number of processes on the single heterogeneous node leads to a suitable load balancing. The strategy applied to determinate the number of processes for each node may consider the strategies described till now, or could be based on simpler considerations. The exploitation of multiple processes for each node has been used successfully in the execution of ScaLAPACK on heterogeneous resources [31]. An advantage of this strategy is that it does not require code modifications, but only the definition of a function to obtain a suitable scheduling, actually a suitable spawn of processes on the single node.

The scheduling function is executed before the application, during the initialization phase, and it requires the size of the data set, the relative velocity V_i for each node, and gives back the global number of processes to execute the computation and the number of processes on the single node.

In PIMA(GE)^2 Lib we consider a trivial heuristics, our aim is not achieve the optimal solution but to obtain an improvement in the exploitation of heterogeneous resources. For this reason we a apply a simplified strategy in order to spawn a number of processes on each node that is proportional to the relative velocity V_i of the node.

The computation is organized as follows: the benchmarking function is run in order to derive the relative velocity V_i of each node. The global velocity of the heterogeneous resource V is obtained as the sum of the V_i, i.e. $V = \sum_i V_i$. The benchmarking results are stored in a specific database, thus they have to be calculated only once on a specific platform, and updated only if the platform is upgraded.

Once known the V_i values, the scheduling algorithm can start. Its aim is to calculate the global number of processes to spawn to effectively exploit the heterogeneous resource, and tune the number of processes on each node efficiently to ensure load balancing. Considering the V_i values and the global velocity of the resource V, the scheduling algorithm consists in a trivial proportion among these values in order to determine the minimum number of global processes to consider and how to distribute them on each node. The result of the scheduling algorithm is a file containing these values.

This phase is hidden to the users, indeed we defined the command `execLib`, that the users employ to execute the application. The `execLib` embeds the execution of the policy already described. The `execLib` compares the V_i values for each node of the cluster; if $V_i \neq V_j$ the `execLib` calls the scheduling function. Thus the `execLib` actually executes the application through an command `mpiexec`, that takes into account the file resulting from the scheduling algorithm to spawn the suitable number of processes on each node.

We provide an example to outline the improvement of the execution time with this strategy. The test considers an heterogeneous cluster of eight nodes, four of them are equipped with a 1,66 GHz Pentium IV processor and 1 GB of RAM, the other two with a Dual 3,0 GHz Xeon with 4 GBs of RAM. The nodes are linked together with a Gigabit Ethernet network. We call *nodes A* the first ones and *nodes B* the second ones; it is clear that nodes B are faster than nodes A.

Firstly we consider the execution of a complex application on the cluster without considering the scheduling policy; the application is the edge detection described in Section 3.5. In this case, the execution time suffers from the difference in computational power of the nodes. Considering the performance obtained with six MPI parallel processes, the execution time is about 251 seconds, however we monitor that the nodes B terminate the actual execution in about 206 seconds, and they are idle until the end of the computation. We would like to reduce this gap between the effective use of each node.

We fixed as reference node one belonging to the nodes B, and we calculate the relative velocity V_i of the other nodes through the benchmark function. We obtain that the velocity of nodes A is about $V_A = 0.84$. We apply the simple scheduling policy and we obtain as proportion that the 54% of the computation has to be assigned to the nodes B, and the remaining part of work load, i.e. the 46%, has to be assigned to the nodes A.

A reduction of the outlined gap could be obtained executing the algorithm on twenty-five MPI processes, since we prefer to consider the minimum number of process to execute the

computations. Fourteen processes are assigned to the nodes B and eleven processes are assigned to the nodes A. In this condition the execution time is about 223 seconds, thus the strategy leads to an improvement in the execution time.

Furthermore, the application of this simple strategy if useful for the future directions of PIMA(GE)^2 Lib, i.e. its integration in distributed and Grid environments. Considering classic distributed environment, this strategy results almost the same, while on the Grid, we use the scheduling function during the resource selection and the execution of the application on such resources; these aspects are described in Chapter 6.4.1.

3.4.2 Building Block sequence optimization

Till now, we have defined an efficient *atomic* parallelization of the single image operation class. Applying this policy in a default parallelization, we derive robust and correct executions of image processing applications. However, an application is typically composed of several library operation calls aimed to elaborate one or more images, and a default parallelization does not take into account this situation. Actually, the parallel execution obtained through a default parallelization performs many unnecessary communication steps, see again Table 3.2, even if the single operations have been parallelized in a proper way. For this reason, it is necessary to define further rules aimed to improve a default parallelization and to define an optimized Building Block concatenation.

In Chapter 2, we deeply described the ParHorus library. As already said, we consider it as a model for the analysis of several topics in the image processing domain. In order to optimize the execution of sequence of operation calls, ParHorus applies the Lazy Parallelization, we gave a detailed description of this policy in Section 2.3.7. The main idea of the Lazy Parallelization is to define a valid set of states and transition functions, and to transparently verify if each operation belonging to the sequence call leads to an illegal or incorrect situation. In this case, further data communications are necessary to enable the correct execution of the single operations.

In PIMA(GE)^2 Lib we consider a simpler strategy, we apply the same I/O pattern for each image operation class. In this way we avoid further communications during the computations, but we do not apply the optimal parallelization policy to all image operation classes. Thus to enable an efficient concatenation of operations, we assume that

- the MPI environment to execute the computation has a fixed number of process N. This value is obtained considering the strategy described in Section 3.4.1.1;
- during the execution data are elaborated by these N processes, and no further processes can be added;

These hypothesis are not restrictive, since they describe the general scenario usually adopted for a parallel computation. In this situation, it is reasonable to distribute data, and keep them on the parallel nodes till the end of the computation, otherwise it results in an useless data movement. Each process is responsible for the computation of a piece of the global data for the part of the computation that involves such data. Thus, data are partitioned when images are instantiated, i.e. they are acquired or created. The partial image is assigned to each process belonging to parallel environment that in turn performs the appropriate operations to its own partial image following the users applications.

Appropriate data communications, such as the collection of a specific value or ghost region exchanges of the image, are properly performed following the *parallelizable pattern* defined for each operation, as described in Section 2.3.3. Data are collected when the execution is completed, or because of an explicit operation, such as print or delete operations. An example to intuitively understand the general management of parallel computations is shown in Figure 3.2.

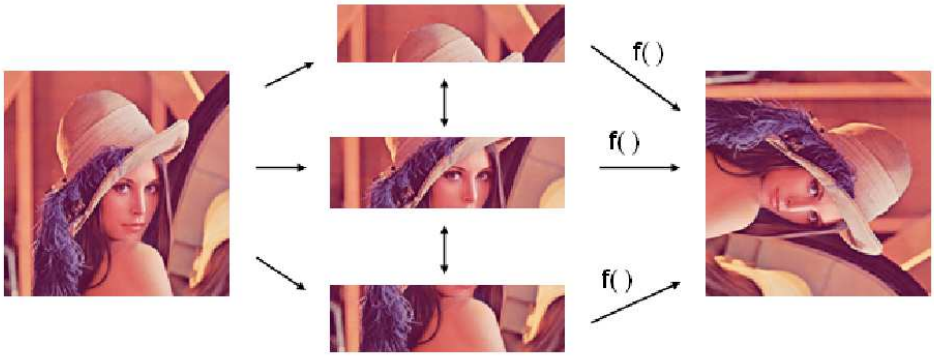


Figure 3.2: *An intuitive example to show how images are distributed, elaborated, and collected among the MPI processes; in this case we considered three processes.*

Applying this policy to the situation described in Table 3.2, we can understand its effectiveness and correctness of this strategy. Considering the schema proposed in Table 3.3, it is to underline the `Import` operation embeds the distribution of the input image, as well as the `Export` operation embeds the collection of the output image.

Comparing the parallelization policy applied in PIMA(GE)^2 Lib with the Lazy Parallelization of ParHorus, we can outline several differences; they mainly involve the data acquisition. In PIMA(GE)^2 Lib, we adopt a parallel approach in the I/O operations exploiting advanced parallel tool, it is deeply detailed in Section 3.4.3.2. It means that in PIMA(GE)^2 Lib each process acquires its portion of data during the I/O operations and there is a direct data distribution during the acquisition phase. Instead in ParHorus it is applied a master-slave approach, that means that the global images are actually acquired

Import(A); UnPixOp(A,B); BinPixOp(A,B,C); Export(C); Delete(A); Delete(B); Delete(C);	Import(A); Scatter(A,locA); UnPixOp(locA,locB); Gather(locB, B); Delete(locA); Delete(locB); Scatter(A, locA); Scatter(B, locB); BinPixOp(locA,locB,locC); Gather(locC, C); Delete(locA); Delete(locB); Delete(locC); Export(C); Delete(A); Delete(B); Delete(C);	Import(locA); UnPixOp(locA,locB) BinPixOp(locA,locB,locC); Export(locC); Delete(locA); Delete(locB); Delete(locC);

Table 3.3: *A sequential application, the corresponding default parallelization, and the parallelization applied in* PIMA(GE)^2 *Lib.*

by a single process, that distributes data to the other processes. Furthermore as already said, in PIMA(GE)^2 Lib we fix the same I/O pattern to distribute data, while in ParHorus the I/O patterns depends on the operations. It is due to enable the suitable parallelization of the algorithmic patterns, for example the Geometric domain operation in ParHorus requires that all processes have the global image. In PIMA(GE)^2 Lib the geometric operation have been implemented in a less efficient way in order to apply the same I/O pattern of the other classes. Our aim was the implementation of an easy parallelization strategy; in fact we avoid the control of the "state" of the data, i.e. if data are suitable partitioned considering the operation to apply, and the consequent data re-distribution. In this way, we avoid both controls and re-distributions. We evaluate the effectiveness of this choice through the executions of data-compute intensive applications, the speed-up values achieved considering PIMA(GE)^2 Lib are satisfactory, as presented in Section 3.5. Furthermore the policy applied to ensure load balancing actually represents an improvement of the parallelization policy applied in PIMA(GE)^2 Lib.

Considering the future directions of the library, i.e. the integration in distributed and Grid environments, the parallel organization of the library enable the achievement of several goals. Maintaining the same degree of parallelism, and avoiding data re-distributions allow the implementation of specific policies to avoid useless data movement in distributed architectures. Furthermore, the policy for load balancing represents a relevant point for a

suitable exploitation of the resources available on the Grid. These topics are mentioned in the remaining of the Thesis.

3.4.3 Data management optimization

Image processing problems lead to data/compute intensive applications, thus a suitable data management really improves their efficiency. We tackle this issue considering effective data acquisition and partition, and we focus on the I/O phases of PIMA(GE)^2 Lib.

Actually, the overhead of the I/O phase has been always considered as a critical point in parallel computing. It is mainly due to a technological gap; in fact in recent years, although great advances have been achieved in the CPU and communication performance of parallel machines, similar advances have not been achieved in their I/O subsystems. However it is possibly to speed-up the I/O phase by using a combination of high-speed I/O hardware, appropriate file system software, appropriate API for I/O in a proper way [110].

Thus our first aim is to allow the PIMA(GE)^2 Lib exploitation of parallel tools possibly present in the execution environment; this actually improves performances of the I/O phase. This goal has to be obtained in a transparent manner, thus also in this case we consider a standard API to hide this step. Furthermore the I/O phase is strictly related to the distribution of data among parallel processes, thus it is important to exploit the connections between the I/O and data distribution to improve the library performance. It implies that the second aim is to consider the suitable combination of reading and distribution policies to speed-up the performance in this phase.

To achieve the first goal we develop the library in such a way that different implementations of the I/O operations correspond to a single interface. The proper implementation is applied depending on the available tools present on the computational resources considered for the PIMA(GE)^2 Lib execution. This consideration is based on the remarks that a parallel application interacts with the underlying I/O hardware infrastructure through a *software stack*, depicted in Figure 3.3. An application developed with PIMA(GE)^2 Lib exploits a software level, which we name *I/O Middleware*, aimed to perform I/O operations. The I/O Middleware interacts with the file system, that in turn effectively exploits the I/O hardware, managing the data layout on disk. Many improvements have been made in the definition of standard interfaces to various layers, both software and hardware; the key point is to enable a proper interaction between the different levels [135].

Starting from these remarks, we develop the I/O operations of PIMA(GE)^2 Lib enabling a proper interaction with wide-spread tools. The I/O functions have been implemented considering MPI, and the exploitation of local, parallel or shared file systems, such as PVFS, Parallel Virtual File System [99], or NFS, Network File System [130].

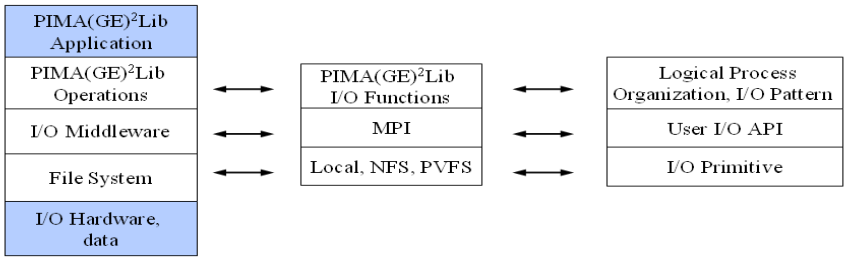


Figure 3.3: *Software stack for parallel image processing applications. Starting from the left, it is explained how an application interacts with data and I/O infrastructure, the tools used in* PIMA(GE)^2 *Lib, and the roles of each software level.*

Another key topic in the achievement of this goal is to evaluate and study the alternative solutions that is possible to consider. In fact, through the I/O operations of the library and the I/O Middleware, the logical organization of the parallel processes and the I/O pattern to access data are imposed; for example the parallel processes could be organized in a master-slave fashion and the I/O pattern could be a cyclic block partition. Thus the analysis of the scenario, and the test of several possibilities lead to the adoption of the strategy that better exploits the underlying I/O subsystem. Therefore we performed a case study about I/O organization in image processing, comparing the a master-slave approach and a parallel I/O. It is detailed in Section 3.4.3.2. This step provides the guidelines to implement the library code.

The second aim to implement an efficient I/O phase is to outline the suitable combination of reading and distribution policy. The key topic in this phase is to consider suitable data distribution pattern, thus we evaluate different possible I/O patterns to derive the one that better fits with image processing. It is detailed in Section 3.4.3.5. Also this experimentation is aimed to derive the guidelines to design and implement efficient code. During both experimentations, we considered the exploitation of advanced parallel tools, therefore the interaction with two widespread parallel and shared file systems, PVFS and NFS, both open source. A brief description is given in the next Section.

As result of the preliminary studies, we adopted the MPI 2 parallel I/O, and a block partition as distribution data pattern, since the low level image processing operations do not require a more complex data distribution. MPI2 combined with PVFS 2, outperforms the other possibilities, providing a reduction of the I/O cost for parallel image processing applications. More in general a parallel I/O approach is effective in the image processing domain, also when MPI2 is combined with a shared file system as NFS. The I/O operations of PIMA(GE)^2 Lib transparently exploit these file systems in order to improve the performance of data acquisition, [29].

3.4.3.1 Experimental conditions

Let us provide few concepts about the software tools used during this work, two widely used file systems for file sharing using clusters NFS and PVFS 2, and MPI-IO.

NFS was designed to provide a transparent access to non local disk partitions. It means that, if we consider the machines of a local area network mounting NFS partition on a specific directory, they are able to share and access all the files of that directory. Therefore a file local to a specific machine, looks to the users of all the machines of the network, as if the file resides locally on each machine.

PVFS stripes files across multiple disks in different nodes of a cluster. It allows multiple processes to access the single part of the global file that have been spanned on different disks concurrently. We considered the second version of PVFS, (PVFS2), that allows the exploitation of fast interconnection solutions and the minimization of bottleneck due to the retrieving of metadata regarding the file to acquire or produce.

MPI-IO permits to achieve high performance for I/O operations in an easy way. Indeed it enables the definition of the most common I/O patterns as MPI derived data types, and in this way permits parallel data accesses. We considered the use of ROMIO [109, 135], a high-performance, portable implementation of MPI-IO distributed with MPICH. ROMIO contains further optimization features such as Collective I/O and Data Sieving. These aspects have been implemented for several file systems, including PVFS2, and NFS.

Tests have been performed on a Linux Beowulf Cluster of 16 PCs, each one equipped with 2.66 GHz Pentium IV processor, 1 GB of Ram and two EIDE 80 GB disks interfaced in RAID 0; the nodes are linked using a dedicated switched Gigabit network.

3.4.3.2 I/O operations

In this Section we present a case study about the I/O operation organization for the image processing operations. To achieve this goal we compared a master-slave approach implemented with MPI 1, and a parallel I/O using the functionalities of the MPI-IO provided by MPI 2. In both cases we tested the interaction with the most common open source file systems for parallel and distributed applications, that are PVFS and NFS. In this phase we fixed the I/O pattern for the data distribution, in this way we focus the interest on the logical organization of the parallel processes. The data distribution pattern is investigated in Section 3.4.3.5. In the next Sections we detail the process organization compared during the I/O phases, and we comment the results of the tests.

3.4.3.3 Different approaches for I/O operations

Typically image processing applications acquire and produce data stored in files. It is essential the effective exploitation of the software stack depicted in Figure 3.3 during these phases.

In order to avoid many small noncontinuous accesses to the disk made from multiple processors, the classic logical organization for data distribution in parallel applications is the master-slave one. It means that a process, the master, entirely acquires data and

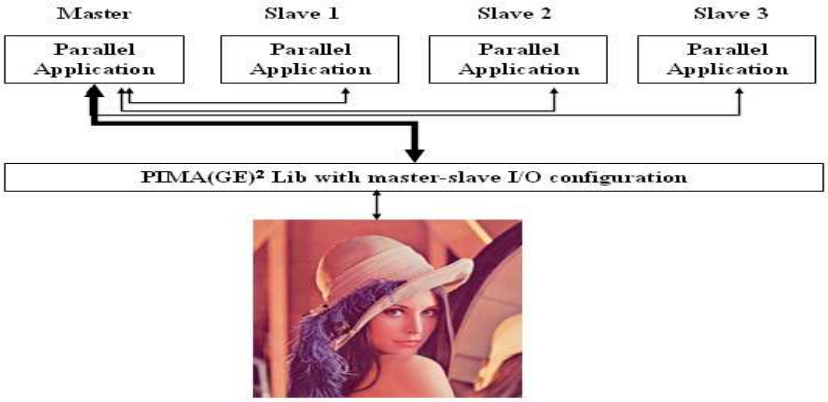


Figure 3.4: *Master-slave approach in accessing data. Only the master accesses the file, and sends portions of the image to the other processes.*

distributes them among the other MPI processes, the slaves, according to the I/O pattern. A specular phase of data collection is necessary for the output operations. Therefore the master is in charge of collecting/distributing data and performing I/O to a single file through a sequential API. This behavior is depicted in Figure 3.4.

However the data collection on a single process results in a serialization of the I/O requests and consequently in a bottleneck, because of the time the master spends in loading the entire data set and in sending the partial data to each process. A specular situation occurs for the Output operations. The waiting time for the distribution or the collection of data increases with the data set size.

An alternative approach to perform I/O operations and avoid unnecessary data movements is provided by a parallel access to the data file. It means that all processes perform I/O operations, but each of them acquires or produces its specific portion of data. The situation is represented in Figure 3.5.

However the partial data of each process may correspond to non contiguous small chunks of global data; it implies that each process accesses the I/O file to load small pieces of information non contiguously located. This situation worsens the performance even if

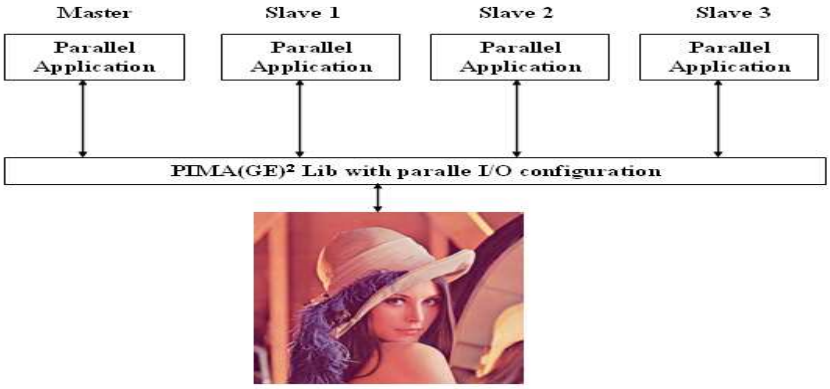


Figure 3.5: *The parallel I/O approach in accessing data. All the processed access data, and acquire their portion.*

sophisticated parallel file systems are used. In fact it is not possible to exploit their optimization policy, since it is mainly designed to efficiently support the parallel access to large chunks of a file or of different files.

Another important point is given by the mismatch between the data access patterns and the actual data storage order on disk. This situation could worsen if data are distributed on different machines. In this case, chunks of data requested by one processor may be spanned on multiple I/O nodes; or multiple processors may try to access the data on a single I/O node, suffering from a long I/O request queue.

However combining the use of proper file systems, and API for I/O in a suitable way, avoids such situations and enables to effectively exploit a parallel I/O. In fact in this way it is possible to span data efficiently and remove the long queue in data acquisition.

3.4.3.4 I/O Experimental results

The comparison between the master-slave approach and the parallel I/O was developed considering a medium size data set, the Computed Tomography (CT) scan of a Christmas Tree (XT)[1], and a quite large size data set, the CT scan of a Female Cadaver (FC)[2]. We fixed as I/O pattern a block partition; for the XT data set, the partial image size varies from 499,5 MB to 31.2 MB considering respectively 1 and 16 processes; for the FC data

[1]The XT data set was generated from a real world Christmas Tree by the Department of Radiology, University of Vienna and the Institute of Computer Graphics and Algorithms, Vienna University of Technology,

[2]The FC data set is a courtesy of the Visible Human Project [138] of the National Library of Medicine (NLM), Maryland.

set varies from 867 MB to 54,2 MB in the same conditions. In particular we measured how the growth of the number of processes impacts on the execution time in each case.

The master-slave approach

We implemented the data partition through a sequential read operation performed by the master that immediately after scatters partial data to the slaves. The execution time (in seconds) are presented in Figure 3.6(a) for XT and in Figure 3.6(b) for FC data set.

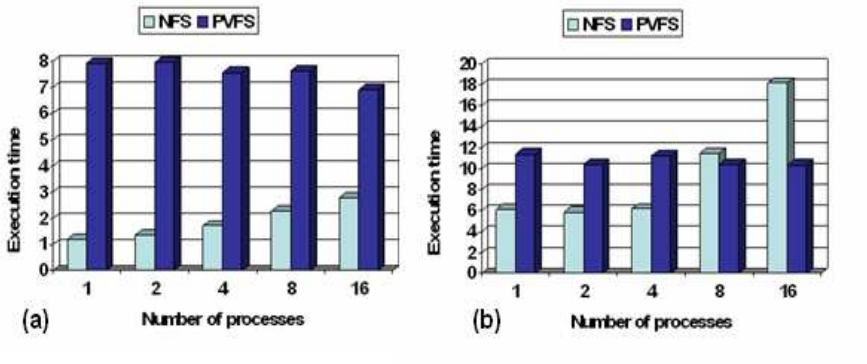


Figure 3.6: *The execution time (in seconds) of the I/O operations using the master-slave approach on the XT data set (499,5 MB) (a) and the FC data set (867 MB) (b).*

In the master-slave approach, the striping of the data among multiple disks obtained with PVFS represents a drawback. Indeed in Figure 3.6 we can see that on both data sets, the I/O performance do not scale and the execution time is almost constant. Both results are due to the overhead related to the use of the file system to access remote part of data.

On contrary, the use of a single disk through NFS represents on average the best solution, despite the data set size may really affect the performance. As it is possible to see in Figure 3.6, the use of NFS performs better than PVFS when we consider a medium size data set; but when we manage a large data set the use of PVSF leads to better performance when the number of process increases.

Parallel I/O

We implemented the parallel I/O using the collective I/O features and the derived data-types provides by MPI 2. The execution time (in seconds) are presented in Figure 3.7(a) considering the XT data set, and Figure 3.7(b) for the FC data set. We can see that the parallel I/O combined with the use of PVFS outperforms the use of NFS and both master-slave solutions.

In Figure 3.7, we can see that MPI 2 and PVFS scales well with the number of processes, since in this case we effectively exploit data access to multiple disks. Instead it does

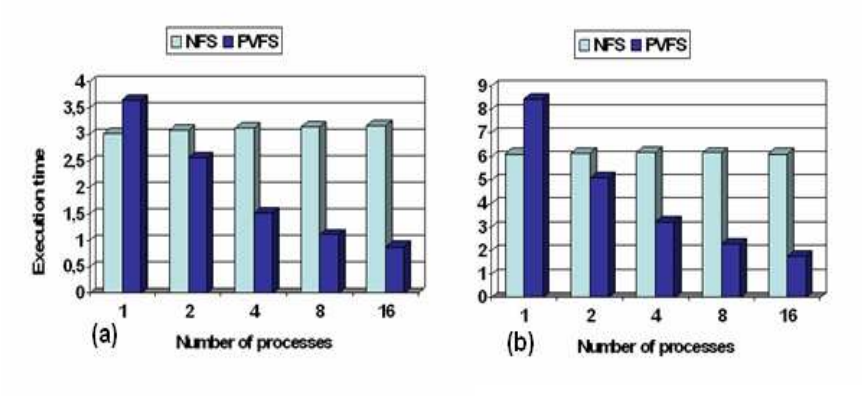


Figure 3.7: *The execution time (in seconds) of the I/O operations using the MPI2 functionalities on the XT data set (499,5 MB) (a) and the FC data set (867 MB) (b).*

not happen using NFS, since NFS was not designed for parallel applications that require concurrent file access to large chunk of files. Therefore as the number of processors and the file size increase, the use of NFS leads to an important bottleneck due to the serialization of the I/O operations. Actually the execution time is almost constant, although the number of processes increases.

3.4.3.5 I/O distribution patterns for parallel image processing

Adopting a programming paradigm Single Program Multiple Data, SPMD, it is necessary to distribute properly the global data among the different processes in order to enable the computation. The aim is to subdivide the whole work into many partial subtasks taking carefully into account the load balancing of the global computation. It means to distribute the work among the parallel processes in such a way aimed to get optimal resource utilization, and avoid bottleneck due to process synchronization. A notable example in literature is given in ScaLAPACK; in this library a complex data distribution pattern is considered in order to achieve the mentioned goals. However in general, the distribution pattern to apply mainly depends on the nature and the regularity of the considered application.

In PIMA(GE)^2 Lib data are partitioned with the same granularity among parallel processes, the number of process is calculated considering the policy described in Section 3.4.1.1. However, the partial data dimension may vary from process to process, since it has to consider a distribution that should be not entirely uniform. This situation is present when the number of parallel processes is not a multiple of the data dimensions.

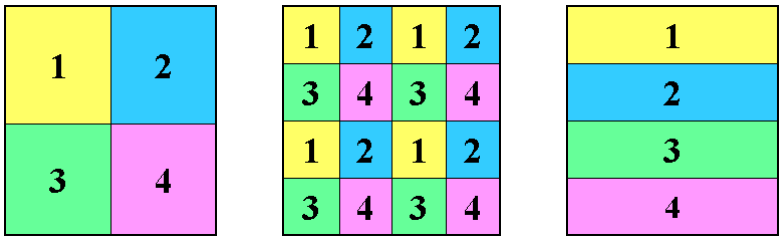
With this background, we started a case study to analyze the possible I/O patterns to apply in the data distribution, in order to understand which is the one that better fits

the requirements of image processing applications. Starting from the results of the Section 3.4.3.2, we fixed the parallel I/O as the logical organization of the MPI processes in the I/O operations, and we compared three different patterns, generally adopted in literature. Since the low level image processing operations do not require complex I/O distribution patterns, we adopt a simple distribution pattern i.e. row block partition. Actually, it speeds the distribution time and also the possible further communications required during the computations. Moreover the outlined I/O pattern ensures load balancing if we consider homogeneous cluster.

3.4.3.6 Different I/O distribution patterns

Considering a 2D data set, we compared three I/O distribution patterns: row and column block partition, cyclic row and column block partition as applied in ScaLAPACK, row block partition. Figure 3.8 depicts these I/O patterns considering a 2D data set, and four processes for its distribution. When we consider 3D data set such patterns become: 3D block partition, cyclic 3D block partition, slice block partition. During the Section we adopt the terminology of the 2D case, because it is more intuitive. As outlined in the I/O phase, the major drawbacks are that the partial data of each process corresponds to non contiguous small chunks of global data, and the possible mismatch between the data access patterns and the actual data storage order on disk when data are spanned on different machines.

In order to overcome these drawbacks, we implement the data partition considering the advanced functionalities of MPI2, in particular we implemented the parallel I/O using the collective I/O features and the derived data-types provided by MPI 2. Also in this case we test the interaction of MPI2 with widespread file systems, as NFS and PVFS.



(a) row and column (b) ScaLAPACK (c) row

Figure 3.8: *I/O patterns considered during the implementation of* PIMA(GE)^2 *Lib. They are depicted for 2D data set, and four processes.*

By using the MPI derived data-types, the file data is seen as a 3D array, and the part

requested by each process as a sub-array. When all processes perform a collective I/O operation, a large chunk of contiguous data file is accessed. The ROMIO implementation on PVFS2 is optimized to efficiently fit the I/O pattern of the application with the disk file striping. Thus the possible mismatches between the I/O pattern of the application and the physical storage patterns in file are minimized. Instead this strategy is not possible considering NFS, because it is a shared file system, thus it enables a simpler access to data, and it does not partition data among the local file system of the parallel nodes. However MPI-IO supports also the an efficient use of this file system.

The distribution phase is made during the I/O operations, to obtain the different I/O pattern it is necessary to change the view of the data file, using the MPI-IO functionalities.

3.4.3.7 I/O pattern experimental results

The comparison between the different I/O distribution patterns was developed considering the same condition presented in Section 3.4.3.2. Therefore, we considered a medium size data set, the Computed Tomography (CT) scan of a Christmas Tree (XT), and a quite large size data set, the CT scan of a Female Cadaver (FC). For each I/O pattern a block partition, the partial image size varies from 499,5 MB to 31.2 MB considering respectively 1 and 16 processes for the XT data set, and from 867 MB to 54,2 MB for the FC data set in the same conditions.

The use of PVFS overcomes the other possibility because of its support to parallel accesses to the data sets by the stripping of files across multiple disks in different nodes of a cluster. In Figures 3.9, 3.10, 3.11, the comparison of NFS and PVFS is depicted considering respectively row and column block, ScaLAPACK, row block I/O patterns. As it is possible to see, the simpler distribution pattern, i.e. the row block partition, achieves the best performance. It it due to the fact that it requires less data access, and to the fact that we tested the patterns on 3D data sets are produced through scanning devices that store data by row, thus the row block partition acquires contiguous blocks of data. For the same reasons the ScaLAPACK partition obtains the worst performance. However, the combination of the two-phase I/O strategy adopted by the collective I/O operations, and the striping of the data among multiple disks performed by PVFS could really limit this gap in the performance of the I/O patterns. The regular nature of the low level image processing operations makes quite useless to apply a complex data partitions. Furthermore some image operation classes require the exchange of data during the computation, thus the ScaLAPACK and the row and column I/O patterns lead to a larger amount of data involved in the communications, and to the definition of further data structure to execute them efficiently. For these reasons, we avoid their use and we apply the row block I/O pattern in the implementation of PIMA(GE)^2 Lib. Moreover the selected I/O pattern ensures load balancing if we consider homogeneous cluster.

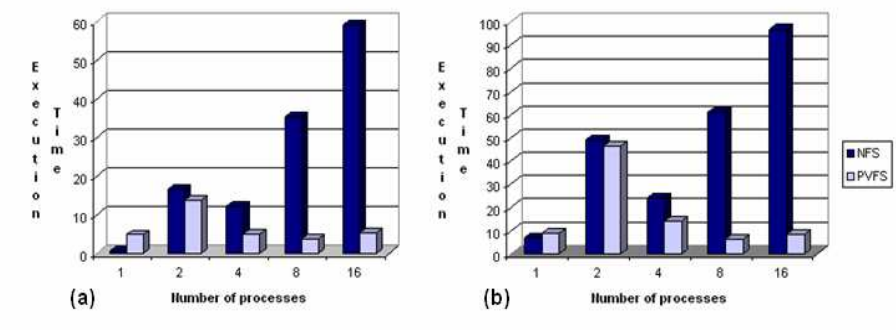


Figure 3.9: *The execution time (in seconds) of the row and column block I/O pattern using the MPI2 functionalities on the XT data set (499,5 MB) (a) and the FC data set (867 MB) (b).*

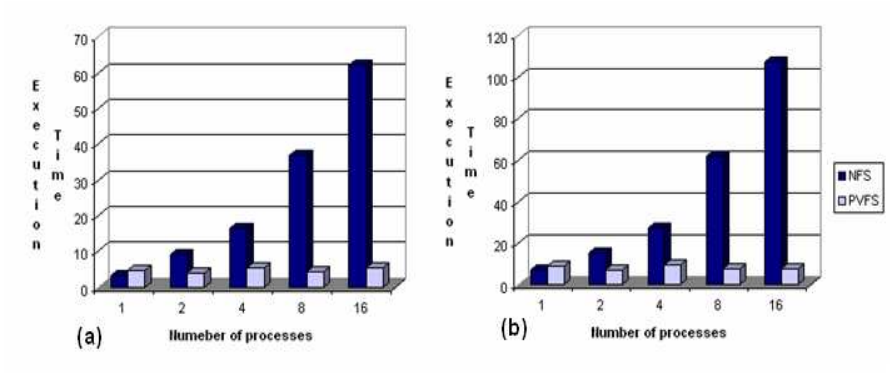


Figure 3.10: *The execution time (in seconds) of the ScaLAPACK I/O pattern using the MPI2 functionalities on the XT data set (499,5 MB) (a) and the FC data set (867 MB) (b).*

3.5 Case study: TMA image analysis

In this Section, we present an example of use of PIMA(GE)^2 Lib, in order to provide an effective application code and evaluate the speed up value that can be achieved using the library. We consider a bioinformatics application, the analysis of the images obtained considering the Tissue MicroArray (TMA) technology. The TMA image analysis is considered as case study also in the integration of the library in distributed and Grid environments. In fact the Grid represents an effective architecture to address some of the issues related to the TMA technology. In the reminder of the Section we introduce this technology and its requirements, we present a restricted number of operations for TMA image analysis developed through the library. We close the Section providing an example of TMA analysis

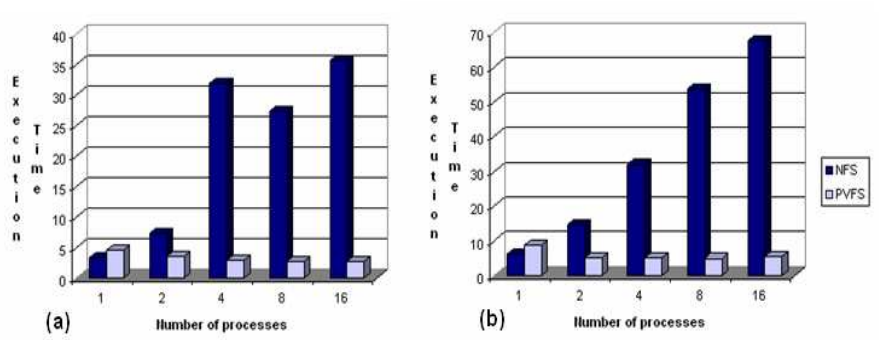


Figure 3.11: *The execution time (in seconds) of the row block I/O pattern using the MPI2 functionalities on the XT data set (499,5 MB) (a) and the FC data set (867 MB) (b).*

performed through PIMA(GE)^2 Lib. The use of PIMA(GE)^2 Lib is can be considered as the first step to develop effective bioinformatics tools.

3.5.1 TMA technology

In recent years, high throughput technologies have increased their potentials and now they are able to produce large amount of genomic, transcriptomic and proteomic data. Nevertheless, most of the molecular-biology methods are scarcely accurate and outputs need to be screened and validated by specific techniques. TMA experiments represent a good validation for data generated following other biotechnological methods, in particular gene expression microarray data. Actually, the major limitations in molecular clinical analysis of tissues include the cumbersome nature of procedures, limited availability of diagnostic reagents and limited patient sample size. TMA technique was developed to address these issues. TMA technique can evaluate the presence of a single biological entity (i.e. the expressed gene) in a parallel way on hundreds of different tissues included into the same paraffin block [86, 88].

More specifically, TMAs consist of paraffin blocks in which up to 1000 separate tissue cores are assembled in array fashion to allow simultaneous histological analysis. In Figure 3.12 examples of TMA images are provided. To obtain TMA paraffin blocks, a hollow needle is used to remove tissue cores as small as 0.6 mm in diameter from regions of interest in paraffin embedded tissues such as clinical biopsies or tumor samples. These tissue cores are then inserted in a recipient paraffin block in a precisely spaced, array pattern, as depicted in Figure 3.13. Sections from this block are cut using a microtome, mounted on a microscope slide and then analyzed by any method of standard histological analysis. Each microarray block can be cut into 100 500 sections, which can be subjected to independent tests. Tests

commonly employed in tissue microarray include immunohistochemistry, and fluorescent in situ hybridization. Tissue microarrays are particularly useful in analysis of cancer samples. Indeed, in this way researchers can check gene expression microarray output, using probes

(a) (b)

Figure 3.12: *Two examples of TMA images; in Figure 3.12(b) the possibility of consider different tissue is underlined.*

to highlight results directly on tissues and validating the different quantity of genes or proteins in case and control samples. Thus, TMA technique is commonly used in many institutes of pathology because it allows the execution of tissue analyses with a decrease in costs and time, and leads to a statistical enrichment of biological profiles.

3.5.2 Issues related to TMA technology

The major issues arising in the analysis of TMA image are storage availability, secure data sharing and image processing.

In fact, from each TMA block it is possible to obtain hundreds of slices that can be processed with many different reagents. It produces many treated spot views to be electronically acquired, and institutes of pathology have to handle a large number of high resolution images. This leads to the necessity of storage capability.

Furthermore, pathologists and researchers working with biological tissues, and in particular with the TMA technique, need a large number of case studies to formulate and validate their hypotheses. It is worthwhile to notice that raw data resulting from TMA analysis are not fully informative. They have to be associated to descriptive metadata, in order to provide classification information. This leads to an effective and secure data sharing.

Moreover, a tool for image processing has to be offered to detect pathologies from digitalized images.

Considering PIMA(GE)^2 Lib, we address the issue of efficient analysis of TMAs; in particular, we developed a restricted number of operations through the library for TMA image

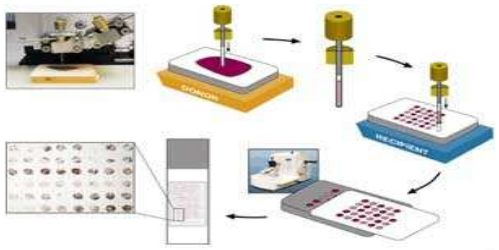


Figure 3.13: *TMA acquisition process.*

analysis. In the remaining of the Section we present the operations and the speed-up values achieved in the analysis of a TMA image.

The further issues already mentioned have been addressed exploiting the Grid, see Chapter 6 Actually we develop TMAinspect, a Grid portal that enables the use of the aforementioned set of PIMA(GE)^2 Lib operations on the Grid, and addresses the other topics.

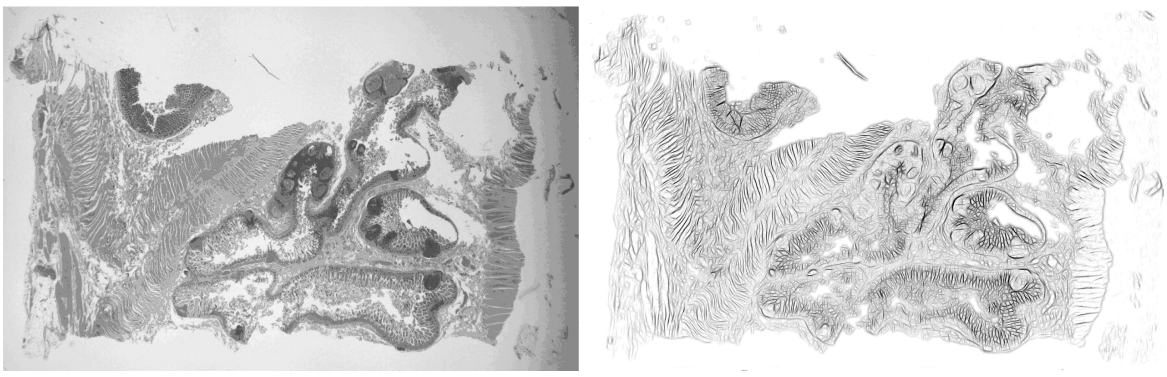
3.5.3 PIMA(GE)^2 Lib for TMA analysis

We provide a set of high level functions properly combining some original library operations for TMA analysis. These operations represent a specialization of PIMA(GE)^2 Lib for the analysis of TMA images. This set of operations includes:

- an edge detection function, extensively used in the analysis of biomedical images;
- a threshold function, which takes an intensity level as input and retrieves a binary image where it is possible to see an area of pixels corresponding to the value;
- a background subtraction function, for removing a constant offset from the signal;
- morphological filters like erosion and dilation kernels, to handle object shapes;
- geometric functions, like rotation, restriction and scale, which allow to better visualize full images or details.

Also in this case, the API of the functions appears completely sequential and the users are not involved in the complexity of parallel programming.

As case study we provide an edge detection applied to a tissue example, in this way we present the performance of the library and the speed-up values that is possible to achieve using PIMA(GE)^2 Lib. The TMA image Figure in 3.14(a), whose size is 1MB, is the input of the algorithm. It represents an haematoxylin-eosin reaction: this coloration differentiates



(a) (b)

Figure 3.14: *TMA image input 3.14(a) and output 3.14(b) of an image analysis, where is applied an edge detection algorithm.*

basic and acid entities in the tissue slice. Basic elements (like cytoplasm) are highlighted in a static grey while acid elements (like nuclei) are in light grey. Figure 3.14(b) represents the result the edge detection.

Processes	*2*	*4*	*6*	*8*	*16*
Speed up	1.8	3.2	4.3	5.0	11.2

Table 3.4: *Experimental results of the TMA edge detection implemented using* PIMA(GE)^2 *Lib. We indicated the speed up values achieved with considering different parallel processes.*

Table 3.5.3 reports the speed up values corresponding to the TMA edge detection using up to sixteen nodes of a Linux cluster. The interconnection is provided by a Gigabit switched Ethernet and each node is equipped with a 2.66 GHz Pentium processor, 1 GB of RAM and two EIDE disks interface in RAID 0. The algorithm requires about 17 minutes to process sequentially the image, while using 16 nodes we can obtain results in less than 2 minutes.

Chapter 4

Moving towards distributed and Grid environments: aims, issues, and optimization policies

In the previous Chapters, we addressed the first goal of this Thesis, i.e. the development of a parallel library for high performance image processing on locally available facilities such as a departmental parallel cluster. In the following Chapters, we address the second goal of the Thesis, i.e. to enable the effective use of the library on distributed architectures and in particular on the Grid. The general framework of this part of the Thesis has been already presented in Chapter 1.

In this Chapter we describe the reasons to exploit distributed architectures, the issues arising in the use of a parallel library in distributed environments, and possible strategies to solve the critical points. The actual tools we developed to enable the execution of parallel image processing applications in distributed and Grid environments are detailed in the next two Chapters.

4.1 Aims and issues

Scientific evolution allows the analysis of phenomena with great accuracy, indeed the production of huge amount of data is obtained exploiting sophisticated instruments, and the study of multidisciplinary problems described through complex models. This leads to data/compute intensive applications, which require storage and computing resources for their solution. The emerging technologies to overcome these requirements are provided by distributed architectures and Computational and Data Grids [52]. Image processing

applications contribute to advances in multidisciplinary research and exhibit similar requirements. Also for image processing distributed and Grid based architectures represent well suited platforms that promise the availability of the required computational power.

In order to develop complex applications, it is necessary to enable the use of parallel libraries as the building blocks to develop applications in a distributed environment. Furthermore it is necessary to exploit dedicated resources, such as a high performance cluster that acts as a server for many remote clients, or shared resources such as computing and data storage devices dynamically available on the the Grid. From the above goals we derive the following issues: interoperability and reuse of high performance code and the suitable exploitation of available resources to enable efficient executions of parallel applications.

To enable code reuse and interoperability, in the recent past different technologies and methodologies have been developed. The generally adopted solution is the legacy code *encapsulation*: the code is left in its environment and dynamically connected to new components through a wrapper allowing the software performs in a Client/Server system and in Service Oriented Architecture [123, 124]. In this way it is possible to exploit the advantages offered by the new infrastructures and to keep alive previously developed softwares, that still provide a useful support to solve problems.

Considering the level of quality and the development cost of parallel libraries, it is of great interest their interoperability and reuse in distributed and heterogeneous environments, as described in Section 2.1. Our aim is to achieve this goal also in the image processing case integrating PIMA(GE)^2 Lib in distributed and Grid environments, enabling an effective use of the library on such architectures. Therefore we evolve the library into new parallel distributed entities, taking into account the second issue already outlined, i.e. the suitable exploitation of available resources to enable efficient parallel executions.

In the following we present two approaches aimed to this general goal. The first approach addresses a client-server interaction in a general distributed environment, while the second approach studies the use of a parallel image processing library in a Grid environment.

In the first case the server provides services that correspond to the execution of library functionalities, possibly at different level of granularity. The server is statically bounded to a fixed set of computing resources, such as a cluster, and is remotely located from its clients. No specific infrastructure is assumed to exist in this case, but we rely on frameworks that enable interoperability in distributed environments, such as CORBA. In the Grid environment we assume the existence of a middleware that implements a distributed virtual infrastructure. In this case it is possible to foresee different situations to enable the use of a parallel library. A remarkable characteristic is the availability of dynamic computing resources.

In both cases, data that have to be processed may in turn be stored on a remote host as well as produced results may be loaded on a different remote destination. Hence remote

data transfer has to be considered. The experimentations made with this aim converge in development of a module for remote I/O, that enriches the efficient I/O operations of PIMA(GE)^2 Lib [29].

In the implementation of parallel distributed entities aimed to image processing, we consider as key points: the library code reuse without a re-implementation or modification to it, the definition of user-friendly tools that still provide the library operations hiding their parallelism, the minimization of the overhead related with the exploitation of distributed environments, the preservation of the efficient parallelism of PIMA(GE)^2 Lib.

4.2 Need for a change in the concept of library

In the era of distributed and Grid computing, the concept of library is evolving. A library is often considered as a static tool, mainly designed for a single user to be executed on an homogeneous architecture in order to develop monolithic applications. Moving towards distributed, dynamic and heterogeneous environments, and to complex multicomponent applications, this kind of interaction with the users appears obsolete and largely unsuited to take advantage of the potentialities promised by the emerging infrastructures.

The first step, to obtain an efficient integration in such architecture, is to evolve the concept of library to a more flexible software entity, that could be considered as a bundle accessed from different users geographically distributed. The new entity can be employed to develop adaptive and distributed applications that are executed in dynamic and heterogeneous environments. The parallel distributed entity can provide the library operations through different kind of interactions depending on the approach followed during the code integration. In this Thesis, we mainly adopted two kind of interactions: a client-sever interaction with the server linked to a fixed set of resources, and a service oriented interaction where services can exploit dynamically available resources, depending on the actual architecture considered during the integration.

Considering a *classic* distributed environment, we apply a client-sever interaction, and we develop a distributed parallel image processing server that provides the library operations. A similar approach has been adopted by several projects, such as NetSolve [1], Ninf-G [132], Nimrod/G [19], NEOS [47]; they are described in Section 5.3. We integrate PIMA(GE)^2 Lib using CORBA, Common Object Request Broker Architecture [33], and in this way we defined PIMA(GE)^2 Server, Parallel IMAGE Processing GEnoa Server [27, 28]; this tool is detailed in Chapter 5. Thus in this case, we evolved the library into a server through the wrapping of the library operations into methods provided by PIMA(GE)^2 Server. It represents an interoperable entity that is able to manage multiple requests performed by different distributed clients, see Figure 4.1. PIMA(GE)^2 Server is coupled with a fixed set of computational resources, such as a specific cluster, that is dedicated to parallel image

processing. We develop I/O functionalities to acquire remote data in order to enable the computation, these functions converged in the I/O module of the library, [56]. The users become CORBA clients, which request the PIMA(GE)^2 Server methods. These requests are executed on the fixed parallel resources on which PIMA(GE)^2 Server runs. At this purpose we assume the use of a local scheduler, such as PBS or LSF, to enable a suitable exploitation of the parallel resources, to accommodate multiple client's requests. This point is not further detailed in the Thesis.

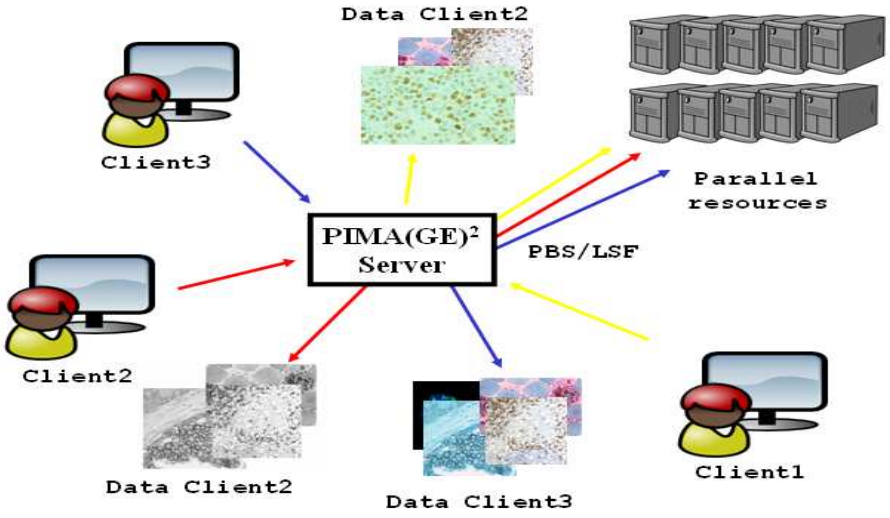


Figure 4.1: PIMA(GE)^2 *Server behavior; it is coupled with a fixed set of computational resources, and is able to manage multiple requests performed by different distributed clients. Also data could be remotely located.*

Considering a Grid environment, PIMA(GE)^2 Lib has been integrated through a service oriented approach, and we exploit the Grid to make different experimentations. Further topics have to be considered on the Grid as for example the application executions, that can be scheduled on several computational resources, as depicted in Figure 4.2. A suitable resources selection may really improve the execution of the applications, as well as the exploitation of the scheduling policy of the library if the applications are executed on heterogeneous resources. We allow this step in a transparent manner or supporting the users in the resource selection depending on the specific experimentation.

First of all we enable the use of the library to develop parallel image processing application and their efficient execution on a Computational Grid, we develop PIMA(GE)^2 Grid, Parallel IMAGE Processing GEnoa Grid tool [30]. It provides a set of services developed using Globus [51] and made accessible through a Java Graphic interface to make easier the interaction with the Grid. The user develops the application considering the operations made available by PIMA(GE)^2 Lib, and then he/she has to provide to PIMA(GE)^2 Grid

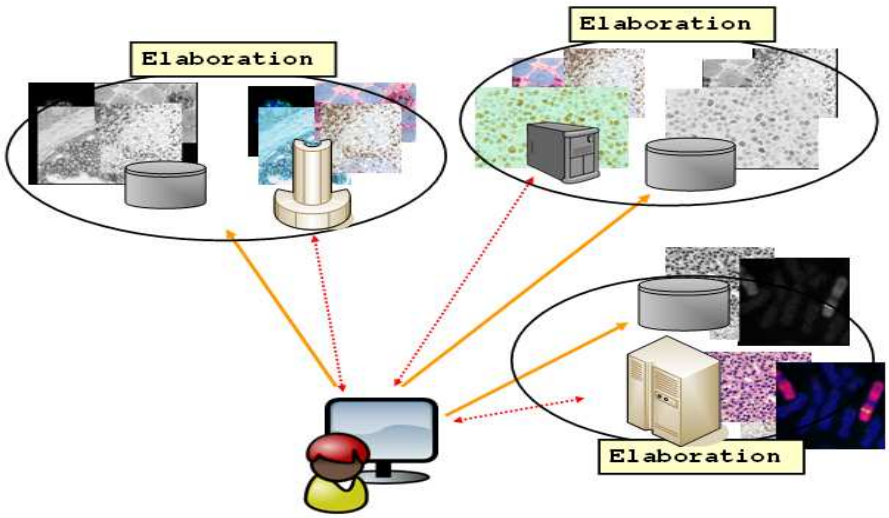


Figure 4.2: *General behavior in a Grid environment, each application can be executed on different resources. Therefore Grid tools are not coupled with fixed resources and can schedule the application executions selecting the computational resources properly. The tools we developed accomplish this task in a transparent manner or supporting the users in the resource selection.*

the source code of the application. The framework discovers the available resources on the Grid, selects the suitable ones; if the selected resources are heterogeneous, it applies the scheduling algorithm of the library. Then PIMA(GE)^2 Grid compiles the code on selected resources, it submits and monitors the job, providing the result back trough the interface.

Furthermore, we consider the use of existent Grid services in PIMA(GE)^2 Lib. We exploit Grid protocol to improve the features of the library considering gLite middleware [60]. We use Grid File Access Library (GFAL) provided by gLite in the data acquisition in order to exploit grid security policies, and thus to comply with possible policy to restricted data access. We develop specialized I/O functionalities, which enables the data acquisition without actually transfer or copy of files from their original positions. Also these functions converged in the I/O module of PIMA(GE)^2 Lib. We avoid a direct use of GFAL by the users [56], actually, the user develops applications through PIMA(GE)^2 Lib employing the opportune protocols to manage data; the execution of the application on the Grid is performed in accordance with a standard EGEE job submission.

Finally we use the library operations to develop *vertical* Grid applications in a specific application domain considering EGEE[1], Enabling Grids for E-sciencE project [48]. We developed a set of specialized functions of PIMA(GE)^2 Lib aimed to the analysis of TMA

[1]An overview EGEE is given in Section 6.3

images, see Section 3.5, we defined a graphical interface to select their parameters, and we exploit further gLite services such as AMGA and GFAL. In this way we define TMAinspect [58], a Grid framework aimed to the TMA images handling. Through the TMAinspect graphical interface the user can select TMA images through a query on metadata, specify the operations to perform and the corresponding parameters, monitor the executions.

In our experiences we evolved a parallel image processing library into a server considering a *classic* distributed environment, and we develop a set of services to use the library on the Grid. In both cases, they simplify the exploitation of the distributed and Grid environments since they provide services that hide the complexity of a direct use of the resources. Moreover, library code reuse and interoperability are ensured, and the integration has been obtained without modifications or re-implementation of PIMA(GE)^2 Lib code. However to enable effective parallel computations in distributed environments, further considerations are necessary. A key point to efficiently exploit a distributed environment is a proper management of data to be elaborated. An actual strategy to achieve this goal is the separation of data acquisition, processing and transfer. This is the topic of the next Section.

4.3 Issues arising in the library integration process

Considering a distributed environment, additional topics have to be discussed to enable an efficient integration and execution of parallel legacy code. We analyse a general scenario for the execution on distributed and Grid architectures of image processing applications in order to focus the main issues arising during the architecture exploitation and the library integration process.

4.3.1 The general scenario

A scientist needs to solve data/compute intensive image processing applications, where data to be processed are located in a remote site as well as the resources to execute the computation. This scenario, i.e scientist, data, and computational resources positioned in different places, reflects the general motivation to exploit distributed architectures, and represents a quite common case. Since the scientist has not the appropriate tool to manage all the aspects involved in such situation, he/she employs a remote tool that offers parallel image processing operations. This situation is depicted in Figure 4.3. Our aim is to develop a parallel image processing tool through a proper integration of PIMA(GE)^2 Lib; it means to enable an efficient reuse and execution of the operations of the library. Considering distributed environments, a key issue is given by an unavoidable overhead related with the data transfer, that worsens the application performance. However it is possible to minimize it though a proper exploitation of the environment and a suitable integration process of

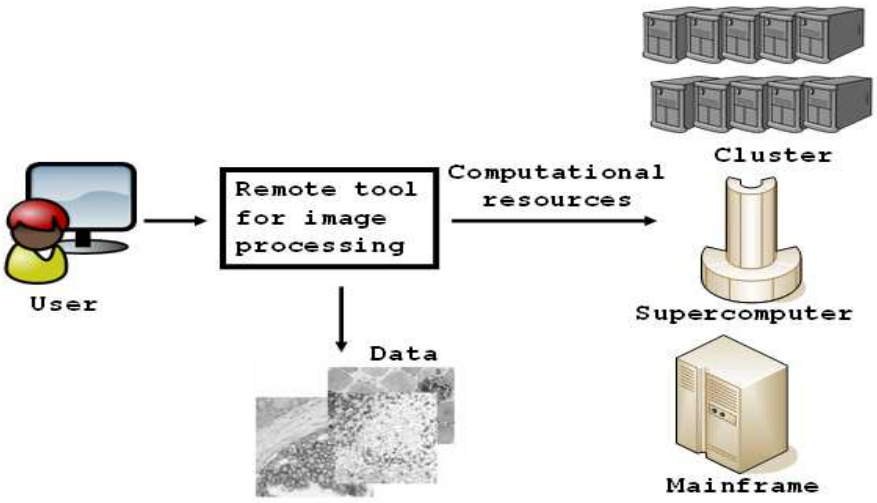


Figure 4.3: *General scenario in distributed and Grid environments. In general scientists, data and computational resources are positioned in different places; a software tool is used to enable the computations.*

the PIMA(GE)^2 Lib operations.

Actually, a *naif* integration of the library could be obtained wrapping each single operation of PIMA(GE)^2 Lib in an operation provided by the tool. Most of the examples of code integration in literature, GrADS [65], MEDLI [69], EasyGrid [17], GEMLCA [37], propose the wrapping of single operation within a method. Those works are discussed in Section 5.3, and 6.2. It is a good way to integrate standalone applications, however, considering a library this approach has to be avoided.

In fact in this case, when the user requires a pipeline of operations, each operation is executed as a single parallel remote computation. Considering a simple analysis, depicted in Figure 4.4, we analyze how it is executed in this condition, and we outline a suitable way to execute it, see Figure 4.5. The execution of a sequence of operations as single parallel remote computation performs poorly mainly for two reasons. The first reason is due to an unsuitable exploitation of the distributed environment, indeed a concatenation of single remote computation implies the addition of the overhead due to data transfer, see Figure 4.5(a). It means a dramatic reduction in the application performance. The second reason is due to an inefficient use of the parallel library, in fact integrating each operation of the library in a single methods, only the optimization of the Building Blocks is applied, and the transparent parallelization policy performs as a default parallelization with consequent useless data communications, see again Figure 4.5(a). This behavior is independent from the approach considered in the integration, i.e. if we consider a server or a set of Grid services, as the problem arises because of the *granularity* of the integration.

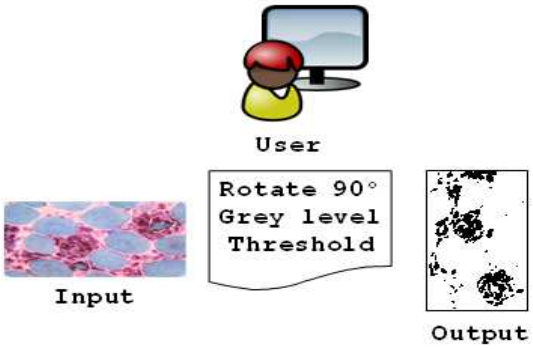


Figure 4.4: *A user requires a simple image analysis. Starting from the input image, the application is composed of three operations: a rotation, a grey level representation, a threshold on a specified value.*

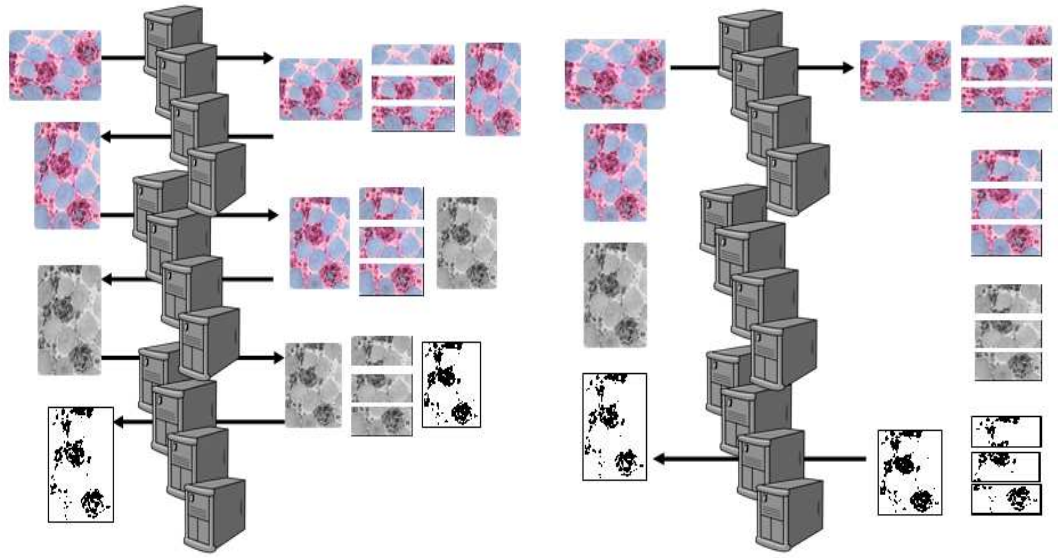
Considering the suitable level of granularity, i.e. the overall application rather than the single operation, the computation is executed as depicted in Figure 4.5(b). It is clear the better exploitation of the distributed infrastructure as well as of the library code.

Therefore it is necessary to analyse additional points during the integration in order to allow effective distributed computations, and preserve the application granularity that is possible to achieve considering PIMA(GE)^2 Lib. This step can bee seen as an *extension to distributed environments* of the PIMA(GE)^2 Lib parallelization policy.

4.4 Enabling efficient executions

Considering the general scenario depicted in Figure 4.3, a further remarkable improvement can be obtained through an efficient data management. Actually it is useful to manage data using proper tools, or a suitable mix of different technologies that combines specific services of distributed and parallel architectures. Therefore we outline three distinct phases in the data management, i.e. movement, access, and processing of data, and we make a clear separation among them during the integration process.

Considering data movements across a distributed architecture, it can efficiently be managed using specific protocols and solutions, such as ftp, http or exploiting Grid protocols for data transfer. Instead during data acquisition the use of advanced parallel tools leads to significant improvements since we avoid the I/O bottleneck and scale on the number of nodes. This situation is depicted in Figure 4.6, where we outline several possibilities for data movement and acquisition. It is important to take advantage of the capabilities provided by the environments.



(a) (b)

Figure 4.5: *The computation depicted in 4.5(a) performs poorly since it requires useless data movement across the distributed infrastructure, and data communication among parallel processes. Considering the suitable level of granularity, the execution of the same computation is represented in 4.5(b).*

With respect to data processing, it is necessary to maintain the parallel library efficiency preserving the granularity of the computations and enabling effective legacy code executions. When it is possible, the scheduling policy of PIMA(GE)^2 Lib could be already exploited. The combination of those technologies leads to a better exploitation of the infrastructure and an effective use of PIMA(GE)^2 Lib operations, it permits to minimize the overhead related with a distributed infrastructure.

Furthermore on the Grid, the separation between data access and processing could be mandatory. Considering EGEE, Grid resources are classified as Computing Element (CE) and Storage Element (SE), see Section 6.3, thus data are stored on the SEs but they have to be processed on the CEs; it implies a data transfer from SEs to CEs. However it is possible to acquire data remotely without transfer data on the CEs where the computation is executed. We experimented the use of GFAL, as described in the previous Section, that enables data acquisition without actual data copy in the Grid environments. It means that in same case, data transfer and acquisition take place at the same time.

In order to separate the three phases of the data management, we need to consider additional hypotheses on the general scenario. We suppose that the scientist does not need to manage or check the partial steps of the computation; this situation reflects the general

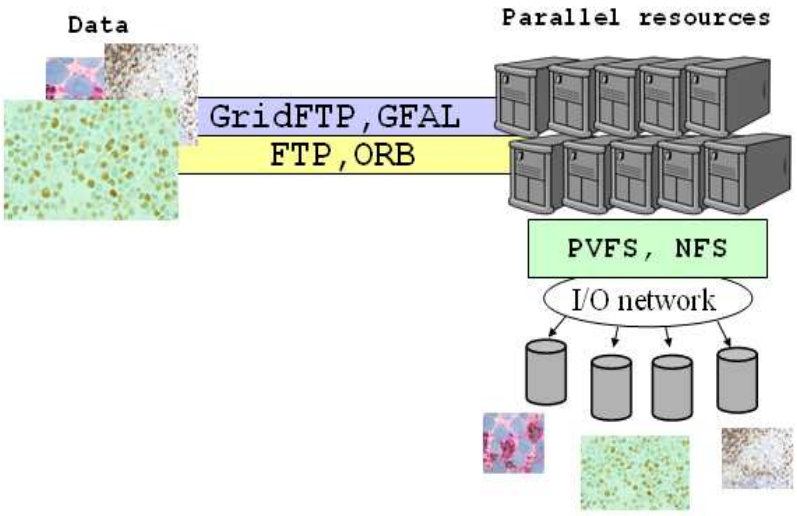


Figure 4.6: *In a distributed environment, it is possible to exploit different protocols to transfer data, and considering parallel machines, they could provide advanced tools to speed up the data acquisition.*

behavior of image processing applications, thus it represents a quite common situation. Therefore the applications can be performed entirely on a remote architecture, and the only output data have to be transfer to the user site. It is reasonable to couple the remote computation to a fixed parallel machine, where it is entirely executed. In a classic distributed environment, the computational resources can be statically fixed; instead on the Grid the resources are selected among those available and could change during the code execution. However, we do not tackle the issue of application rescheduling during the execution since it is out of the topics of this Thesis.

Starting from these remarks, it is possible to transfer data and to execute the computation on the selected/fixed parallel resources. These hypotheses enable a clear separation among movement, access, and processing of data; of course the actual implementation of the single points depends on the specific environments. However this general strategy leads to different advantages:

- an actual data transfer among distributed resources is performed only to move the Input/Output data, this task can be accomplished providing optimized data transfer services;
- since data are moved local to computational resources, the execution of user applications does not involve the data movement in a distributed architecture;

- exploiting a parallel machine we can take advantage of the parallel tools it may provide. As described in Section 3.4.3, during the development of PIMA(GE)^2 Lib, we already considered the use of the best I/O subsystems to enable efficient data acquisitions, thus we preserve a part of the library optimized parallelization policy;

- since the execution involves actual local data on the parallel resource, it is possible to define the suitable strategy to preserve the granularity of the library. The definition actually varies depending on the environments, it may be also possible to exploit the scheduling policy of the library;

This permits a considerable reduction of the overhead due to distributed infrastructures. Actually, avoiding useless data movement during executions and enabling a suitable integration of library code, the overhead is relegated to mandatory steps, such as the start-up of the environments and the data transfer. In this way we avoid the inefficient "naif" integration presented at the beginning of the Section, on the contrary we efficiently exploit the distributed environment, and we effectively reuse library code.

Chapter 5

A distributed parallel image processing server

5.1 Enabling library code interoperability

A great interest in scientific community is toward code reuse and interoperability. Actually, complex problems, also defined as multidisciplinary problems, require the cooperation of applications from different domains, and the solution of the main problem is assembled from the solutions of the single applications [77, 8]. Each single application could be developed using different languages or running under different operating systems; in such conditions their direct interaction could be a hard problem. A suitable solution is to reuse single codes through components or services to enable their interoperability rather than develop from scratch a new monolithic application [1, 63].

In this Chapter we consider the use of component technology in order to enable an interoperable use of the PIMA(GE)^2 Lib operations. The component programming model emerged to facilitate the development of modular software and the interoperability of large-scale software systems [131]. It strongly emphasizes the encapsulation of code into components designed with standard, clearly defined interfaces and connected in a framework. Typically multidisciplinary applications are composed of a large number of complex components; some of them may involve images and require an efficient processing. Our aim is to enable the development of high performance image processing components to be employed in such cases. We start considering a client-server interaction to enable the access to image processing components. The client-server model has become one of the central ideas of network computing; it provides a convenient way to efficiently interconnect programs that are distributed across different locations. Usually in a client-server model, the server is activated and awaits client requests. Clients are geographically distributed

and exploit heterogeneous architectures.

Many mature client-server technologies are already available, they were designed to ensure security, "friendliness" of the user interface, and ease of use. We considered the CORBA framework [33] to develop PIMA(GE)^2 Server, Parallel IMAGE processing GEnoa Server, obtained through the encapsulation of the library operations. In particular we employ the TAO implementation [127] to develop the server. The main difficulty related with the framework is to allow the parallelism of the computation; in fact CORBA is aimed to reuse mainly sequential code, hence it does not support intrinsic compatibility with any kind of parallel environment. The strategy to enable the execution of the parallel computation inside a CORBA object is presented in Section 5.4.3; it has been obtained without modifications to the standard imposed by the Object Management Group, OMG [98]. PIMA(GE)^2 Server totally manages the parallelism of the computation without involving the users. This leads to the definition of a user-friendly tool that combines parallel and distributed computing without further burdens for the users. Similarly to the library case, the Application Programming Interface (API) of PIMA(GE)^2 Server appears totally sequential; it is described in Section 5.4.2.

In the previous Chapter we analyzed general scenario for distributed/Grid processing, and we outlined the overhead introduced in the integration of the library code. To minimize this overhead and to enable a proper encapsulation and an efficient reuse of parallel library code, we considered aspects concerning data management and the granularity of applications. In Section 5.4.4 we describe how we accomplish these tasks in PIMA(GE)^2 Server. Through a suitable data management, PIMA(GE)^2 Server enables the development of distributed heterogeneous applications with a limited overhead as demonstrated by experimental results provided in Section 5.4.6.

5.2 The CORBA framework

CORBA, Common Object Request Broker Architecture, is an object-oriented architecture that enables the creation and the management of distributed program in a network. It allows programs at different locations and developed by different vendors to communicate performing in a client-server architecture. Thus two important features of CORBA are the platform independence and the language independence; it enables code reuse in proper distributed objects and supports application portability and interoperability.

The main element of the architecture is the Object Request Broker, ORB. It actually enables the communications among objects without having to understand their location and implementation. After the ORB is initialized, all CORBA objects can be invoked by applications like local software objects. When a CORBA client requires methods provided by a CORBA server, the ORB has to: intercept a call from the object-client, find the correct

object-server to satisfy the client requests, pass it the parameters, invoke its methods, and return the results of the request to the client. In addition, CORBA automatically applies a range of useful services to enable communications.

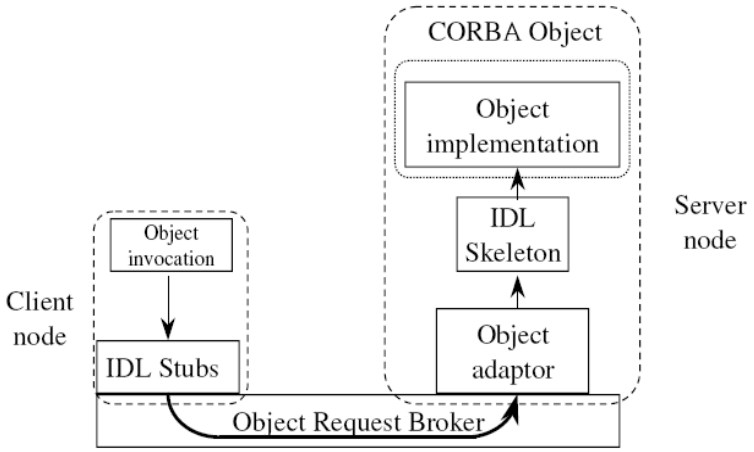


Figure 5.1: *The CORBA client-server architecture.*

To develop objects, CORBA provides the Interface Definition Language (IDL), it is aimed to the definition of the interface of an object, while the implementations is performed using some other language. In this way is possible the reuse of legacy code. Standard *mappings* from IDL to a specific implementation language exist for several widespread programming language. The IDL generates the client stubs and the server skeletons, that are necessary to define the object and to enable the invocation of its methods, as depicted in Figure 5.1.

CORBA uses the notion of object references to facilitate the communication between objects. When a client requests methods provided by a CORBA object, it has firstly to obtain an Interoperable Object References (IOR) for that object. Using the IOR, the client can exploit methods of the server. The ORBs usually communicate using the Internet Inter-ORB Protocol (IIOP). The IIOP supports interoperability of different ORB products and allows the communications among objects developed with different CORBA implementations. The interoperability is depicted in Figure 5.2.

As usually happens in a client-server architecture, the client code has to be compiled and submitted according to the specifications of the CORBA implementation considered.

CORBA Component Model (CCM) [140] is an addition to the family of CORBA definitions. CCM extends the CORBA object model by defining features and services in a standard component environment. This environment enables to implement, manage, configure and deploy components, that are integrated with commonly used CORBA Services. It was

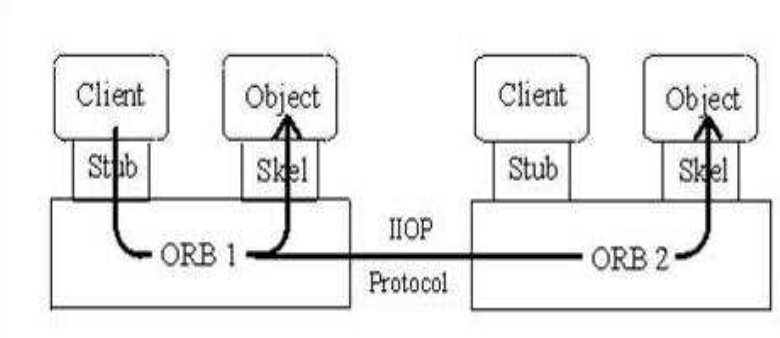


Figure 5.2: *IIOP enables the interaction among different CORBA implementations.*

introduced with CORBA 3.0 specification, the OMG IDL has been extended to express component interconnections. A component can offer multiple interfaces, each one defining a particular point of view to interact with it. CCM has a component container, where software components can be deployed and use the set of services provided.

5.3 Related Works

The exploitation of distributed architecture to develop applications and to allow an interoperable reuse of legacy code is the subject of successful projects, such as Cactus [63] and NetSolve [1]. They provide a framework where users can access both hardware and software resources distributed across a network. Users can also integrate any arbitrary software component as a resource into the systems according to the programming languages supported by the frameworks.

More in detail, Cactus provides a modular and portable programming environment for parallel high performance computing; it can exploit other technologies and programming languages. The name Cactus comes from the design of a central core ("flesh") which connects to application modules ("thorns") through an extensible interface. Thorns could be the user applications, as well as standard computational capabilities, such as parallel I/O, data distribution, or check-pointing. Cactus provides easy access to many software technologies, and enables the development of applications, hiding the underlying architecture.

NetSolve is a client-server system, that considers agents to mediate the interaction among the users, resources and applications. When a user requires the solution of an application, NetSolve searches for computational resources on a network, selects the suitable ones, and solves a problem giving back the answers to the user. Load-balancing and fault-tolerance policies are used to ensure good performance. All these tasks are performed by Agents. The NetSolve project has been extended in the GridSolve project, in order to exploit the

Grid. NetSolve/GridSolve exploit Grid resources and are part of the so-called *Network Enabled Server*, NES, as Ninf-G [132], Nimrod/G [19], NEOS [47]. NESs exploit servers available in different administrative domains through the classical client-server or Remote Procedure Call (RPC) paradigm, they also implement the GridRPC [82]. In NESs, clients submit computation requests to a scheduler whose goal is to find a server available on the grid. Scheduling is frequently applied to balance the work among the servers, and a list of available servers is sent back to the client; the client is then able to send the data and the request to one of the suggested servers to solve their problem.

Considering distributed image processing, it is possible to find different tools aimed to enable distributed processing of images [34, 24, 35, 76]. They enable remote access to distributed data, provide image processing operations, and execute the computations on distributed resources. However they do not tackle an efficient parallel processing, or the integration of legacy code.

The integration of parallel code in a distributed infrastructure has been considered in different works and projects. In particular, the reuse of MPI codes in CORBA has been obtained enabling their execution using external command inside CORBA objects [74, 75].

Furthermore several efforts are aimed to add parallelism in the CORBA framework. Projects as PARDIS [83] and Cobra [105] were first attempts to introduce parallel object in CORBA. They define parallel CORBA objects through new IDL constructions in order to achieve SPMD model. They have a very close approach since both extend the CORBA object model to a parallel object model. A new IDL type is added, describing data type, and size. In PARDIS, data distribution is let to the programmers, it is the main difference with Cobra, where a resource allocator is provided. Also PaCO [107] is an early attempt to create parallel CORBA object. PaCO extends the IDL language with new constructs. These constructs enable the specification of the number of CORBA objects being part of a parallel object and the data distributions of the operation parameters.

PACO++ [101] is the continuation of the PACO project as it shares the same parallel object definition. However, PACO++ represents a portable extension to CORBA, so that it can be added to any CORBA implementation. The parallelism of an object is in fact considered to be an implementation feature of this object, and the OMG IDL is not modified. The general framework to run the PaCo++ object is PadicoTM [38]. It is an open integration framework for communication middleware and runtime. It allows several middleware systems to be used at the same time. Thus PadicoTM enables to simultaneously use of CORBA and MPI.

These projects leads to definition of CORBA-compliant frameworks, and some of them do not respect the standard specifications of OMG. An OMG Request For Proposal was published with the aim to solicit for the creation of a standard CORBA version that efficiently support parallel computing [94]. The proposal accepted was submitted by the

union of several industrial companies, and it proposes to add high performance to CORBA through the definition of a parallel ORB instead of an extended IDL [87].

5.4 PIMA(GE)^2 Server

In Section 4.3.1we already presented the general scenario of a distributed environment and made several hypotheses to enable an efficient reuse of the library code. In this phase, an unavoidable overhead is added to the application performance; it is possible to minimize it through a proper exploitation of the environment and an effective integration process.

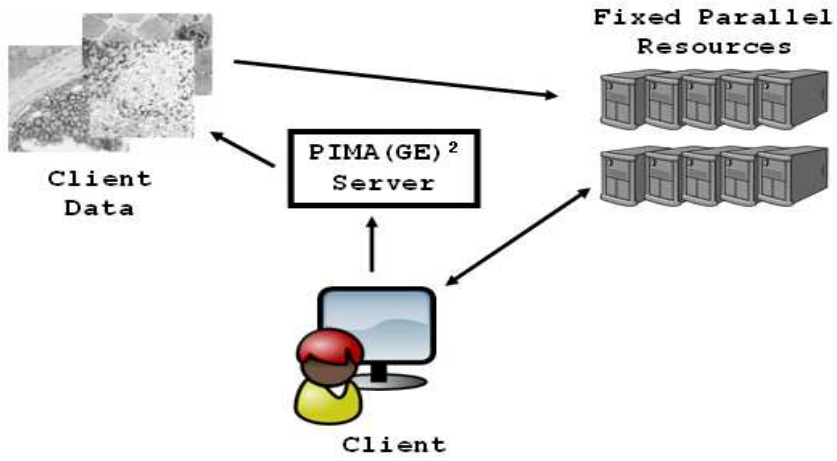


Figure 5.3: PIMA(GE)^2 *Server behavior; it is coupled with a fixed set of computing resources. Data could be located remotely, thus* PIMA(GE)^2 *Server provides methods to specify data location. The images are transferred on the parallel resources where the request is executed and are processed. The output images are transferred in a site specified by the client.*

A notable improvement can be obtained through an efficient data management, thus we outlined three distinct aspects in this phase, i.e. movement, access, and processing of data, and we made a clear separation among them during the integration process. The actual implementation of each single point depends on the specific environment considered during the Thesis; in this work we analyze how to accomplish these tasks in a "classic" distributed environment, using a well-established framework as the standard CORBA.

We assume that PIMA(GE)^2 Server is coupled with a fixed set of computational resources, such as a departmental cluster, where the client requests are executed. Data could be

remotely located; in this case we assume that data are free available and it is possible to download them through an anonymous file transfer. Otherwise if data access are restricted through an authorization policy, we suppose that users are trusted to access and move data across the distributed infrastructure, thus also in this case, users can download data through an anonymous file transfer. In such hypotheses, PIMA(GE)^2 Server is responsible to transfer the images from the site specified by the user to the fixed parallel resource. This scenario is depicted in Figure 5.3. The output images could be possibly transfer in a site specified by the client.

Starting from these remarks we describe the actual implementation of PIMA(GE)^2 Server in the remaining of the Section. Our aim is to minimize the overhead due to distributed architectures and enable an efficient reuse of library code.

5.4.1 Possible approaches

As already said, the integration of parallel code into CORBA objects or components is not a trivial task. Actually CORBA was not originally intended to support parallelism within an object, and there is not an intrinsic compatibility between CORBA and a parallel environment as MPI. In Section 5.3, we described CORBA-based framework developed to overcome this issue, however we prefer to consider a standard CORBA framework in the development of PIMA(GE)^2 Server.

Some CORBA implementations provide support for multi-threading in the execution of objects, as for example the TAO ORB implementation [127]. It means that several processes are able to share simultaneously a physical memory within a single computer. Such level of parallelism concerns the object implementation, thus it does not enable the development of parallel objects, but simply a more efficient execution of standard CORBA objects.

In order to integrate MPI parallel code into CORBA object, it possible to consider different simple approaches. As already considered in literature and presented in Section 5.3, a possible strategy is to enable MPI parallel applications in CORBA through an external commands. The object wrapping an MPI application is implemented using its executable code, i.e. the CORBA object hides a call to the *mpirun* of the respective executable. However considering a parallel library this strategy performs poorly. In fact in this case, the integration of PIMA(GE)^2 Lib has to be obtained through the wrapping of the single operations, it leads to the "naif" integration presented in Section 4.3.1. We already analyzed the ineffectiveness of such atomic integration.

Another possible strategy is to enable the use of parallel code with a CORBA object adopting a master-slave approach. In this case a particular process plays the role of a master that is connected to slave processes exploiting the MPI environment; only the master process is encapsulated in the CORBA object. There are two general drawbacks

in this approach, the first is that it requires the modification of the MPI legacy code if the computation does not follow a master-slave organization. The second one is that the master represents an important bottleneck when two (or more) MPI codes elaborating the same data has to communicate each other. It is due to the granularity of application integrated; in this case when a client requires a sequence of MPI codes, we obtain the same situation presented in Section 4.4, i.e. useless data tranfers are performed for each service. Furthermore, the situation is made worse considering the presence of the master, that actually represents a bottleneck because of it has to take/return data from the ORB and distribute/collect data among MPI processes. Thus, the solution does not scale on the number of process.

However starting from the master-slave approach, it is possible to define a similar behavior to enable the coexistence of CORBA and MPI avoiding the outlined disadvantages. This strategy is presented in Section 5.4.3.

5.4.2 The server API

The first step to allow a simple and coherent use of the library in a distributed environment is the definition of effective and flexible interfaces for PIMA(GE)^2 Server. The API of PIMA(GE)^2 Server has to allow the development of efficient image processing applications in an easy way. The definition of the API can be achieved through a natural evolution from the library code to PIMA(GE)^2 Server.

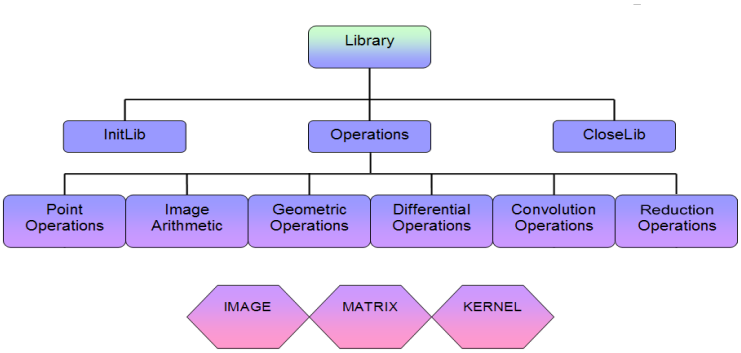


Figure 5.4: *Overview of the* PIMA(GE)^2 *Lib hierarchy of image processing object. We exploit the hierarchy to drive the definition of the* PIMA(GE)^2 *Server interface.*

The crucial element to achieve this goal becomes the exploitation of the classification of the PIMA(GE)^2 Lib operations, and their organization in a hierarchy of image processing objects. This step was already performed to define the API of the library, see Sections 3.3.

This effort is useful to provide the model also for the definition of an adequate PIMA(GE)^2 Sever interface [27].

Starting from the defined hierarchy, depicted in Figure 5.4, the implementation of the API is totally planned and easily obtained. It actually provides a model for the definition of the interface; this structure has been codified using the CORBA Interface Description Language (IDL). Such interface represents the API that will be called by client applications to exploit the methods provided by PIMA(GE)^2 Server. An example of interface of few methods provided by PIMA(GE)^2 Server is given in the box code 5.1, while an example of client code using these methods is given in Section 5.4.5.

Listing 5.1: *An example of* PIMA(GE)^2 *Server API.*

```
1 module LIB {

3 // Definition of data type and data structure
  enum  HxGeoIntType { LINEAR, NEAREST };
5 struct PIXEL { double   x; };

7 interface Operation{

9 // Mandatory methods to initialize and finalize the Server
    void InitLib();
11  long CloseLib();

13 // Local I/O operations
    void readImageFromFile(inout long IdImg, in string fname);
15  void writeImageToFile(in long IdImg, in string fname);

17 // An example of Unary and Binary Operations:
    void unaryPixOp(in long IdImg, in char typeOp);
19  void binaryPixOpI(in long IdImg1, inout long IdImg2, in long typeOp);

21 // Geometrix Operations
    void rotImg(in long IdImg1, out long IdImg2, in double alpha,
23              in IntType gi, in long adg, in pi bg);
  } };
```

The invocation of the methods `InitLib` and `CloseLib`, presented on lines 14-16 of box code 5.1, is mandatory to employ the methods provided by PIMA(GE)^2 Server. As it is possible to see in the same box code, PIMA(GE)^2 Server provides a sequential API that hides the parallelism of the image processing objects and the strategies applied to enable the computation. From the client point of view, the interface looks like a standard CORBA server; the parallelism of the library and policies applied to allow MPI-CORBA compatibility and an efficient code reuse are totally hidden.

5.4.3 Enabling parallel computations in CORBA

The main difficulty arising in the integration of PIMA(GE)^2 Lib is due to the parallelism of the library code. As already described in Sections 5.3 and 5.4.1, several approaches are possible to execute MPI code in CORBA. Starting from the master-slave approach presented in Section 5.4.1, we experimented a different strategy to allow parallel executions in a CORBA object avoiding the outlined drawbacks. The organization of the computation is described in this Section, while specific aspects are detailed in Section 5.4.4.

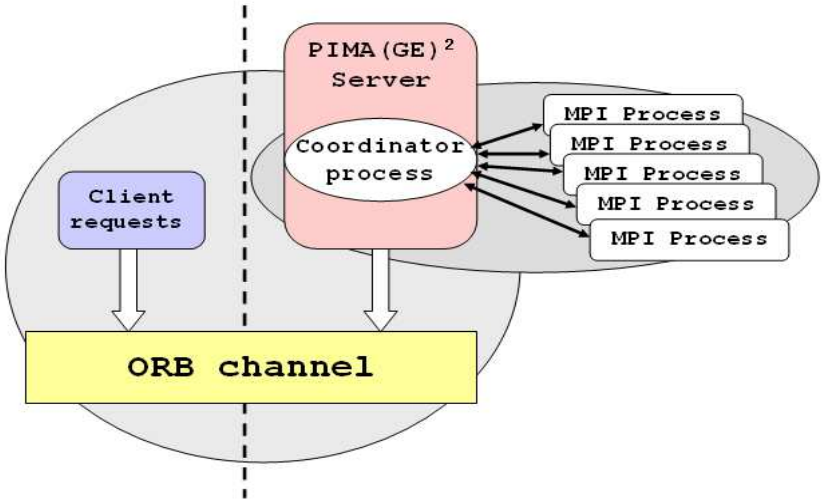


Figure 5.5: *The CORBA-MPI interaction. The coordinator process collects the client requests and returns the images through the ORB. To enable the parallel computation the coordinator communicates the user's requests to the other parallel processes exploiting the MPI environment.*

We implement the wrapping of the library operations as a data parallel application with a coordinator process. The coordinator acts as the gateway that allows a cooperation between MPI and CORBA environments exploiting its dual position in the software architecture, as depicted in Figure 5.5. In fact the coordinator is the only MPI process actually integrated in the CORBA object. It activates the ORB, and acts as a standard CORBA server, i.e. it receives requests from CORBA clients geographically distributed. At the same time, the coordinator is one of the MPI processes spawned to execute the library operations, thus it is able to exploit MPI environment to communicate with the other processes and manage the computation. The coordinator process has a similar role of the master in the approach described in Section 5.4.1, but the exploitation of the coordinator does not imply any modification to the library code. To enable its correct mediation between CORBA and MPI, the implementation of a proper wrapper is necessary; a simplified schema is presented in code box 5.2. The policy applied to avoid also the bottleneck on

the performance is described below.

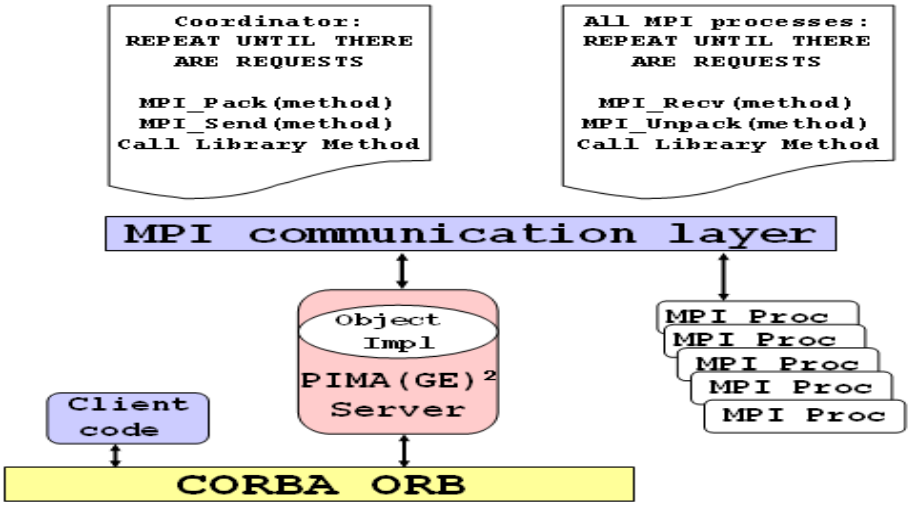


Figure 5.6: *The management of MPI execution inside* PIMA(GE)^2 *Server. For each method required in the client code, the coordinator employs MPI functionalities to communicate proper information to the other parallel processes. It is repeated until the invocation of the CloseLib.*

The computation is organized as follow. When a client requires PIMA(GE)^2 Server methods, the coordinator process performs several steps as, the construction of data structures, the start-up of the MPI environment, the conversion of data types. A MPI communicator context is associated to the specific client code until the end of the computation; it enables a correct computation. The coordinator process performs these steps when the `InitLib` is invocated, indeed its use is mandatory. The `InitLib` invocation also alerts the MPI processes to check for an incoming message, the alert ends only with the invocation of `CloseLib`. It finalizes the MPI environment and the specific session; its use is mandatory as well. The alert is obtained considering a proper conditional structure that is specific for the current client-server session. This behavior is depicted in Figure 5.6.

Within the same client-server session, the coordinator collects the client requests through the ORB and activates the related methods. The activation of the single method corresponds to a send of information in the MPI environment; such information concern the method to run and its parameters. In this phase, each method is labeled through a TAG that is necessary for the MPI process to recognize the specific operation. The coordinator employs MPI functionalities to communicate information, and to label the methods. On the other side, the MPI processes have been already alerted through the `InitLib` invocation, thus they check for an incoming message. In this way they are able to receive the information sent by the coordinator. Through the TAG, they identify the method to run,

and execute the operation with the proper parameters.

Data involved in the computation are managed considering the MPI environment, and they are processed following the parallelization policy applied in the native library. When the MPI execution of the single method ends, the coordinator gives back the information to the client acting as a CORBA server through the ORB channel. However to avoid an useless data movement across the infrastructure and execute application at a suitable granularity level, we organize the execution in the following way. Until the end of the specific client-server session, data are fixed on the computational resources where the server is running and they are maintained partitioned among the MPI process.

Listing 5.2: *A part of the* PIMA(GE)^2 *Server implementation. It is aimed to explain how the coordinator process acts to enable the execution of MPI code in the CORBA architecture. The implementation of a general method is reported in box code 5.3.*

```
1
2 if (myId == coordinator) {

4 //  Only coordinator activates the client requests
    operation_impl->method(parameters);
6
  } else {
8
  while(stop == 0) { // Condition ending with the CloseLib
10
    MPI_Probe(0, MPI_ANY_TAG); // Alert wait a msg
12  switch(status.MPI_TAG){ // TAG to active the proper method
     case 0:
14     // CLOSELIB
       MPI_Recv(proper parameters);  operation_impl->CloseLib();
16     stop = 1; break; // Exit conditions

18   case i:
       // Operation corresponding to the i case
20     MPI_Recv(buffer, MPI_PACKED); MPI_Unpack(parameters);
           operation_impl->method(parameters);
22 }
```

This situation is maintained until the invocation of `CloseLib`, or an operation requiring a gather of data. In this way we avoid also the bottleneck of the master-salve approach and we obtain an efficient implementation of the code. In Section 5.4.4 we detail different aspects of the computation.

Listing 5.3: *The implementation of a general method. Only the coordinator packs the parameters and sends the information to the other MPI processes.*

```
1 //  inside the method implementation
2 //  only the coordinator
  if (myId == coordinator){ // Only coordinator
4   MPI_Pack(methods, parameters); // Pack of the info
    for(aa=1;aa<numproc;aa++)
6     MPI_Send(buffer, MPI_PACKED, MPI_TAG); // Send and TAG
  }
8 // All processes call library
  call LibraryMethod(parameters);
```

This strategy has been obtained using MPI functions, and a proper wrapper in the integration. It does not require modifications to the library legacy code and enables a direct interaction between the client and the library operations [28]. Moreover this strategy leads to a reasonable overhead in the application performance, as presented in Section 5.4.6. It is to notice that the performance of distributed applications already scales on the number of processes. Further details of the strategy are described in Section 5.4.4.

5.4.4 Optimizing code execution

A key issue to develop efficient distributed applications is a proper data management. In fact a clear separation among movement, access, and processing of data enables a better exploitation of the architecture and an effective reuse of the library code. We assume that data are freely available or users are trusted to access and move data across the distributed infrastructure; in both cases it is possible to download them through an anonymous file transfer. Therefore in the implementation of PIMA(GE)^2 Server we consider as key aspects: an efficient data transfer, an effective data acquisition, reduced volume of communications in client-server sessions, and the preservation of granularity of the applications. A proper management of these points actually reduces the overhead of the integration. In order to force this type of interaction among data, clients and PIMA(GE)^2 Server, we arrange the computation in this way:

- when the client requires the processing of remote images, PIMA(GE)^2 Server transparently uses protocols to efficiently transfer data to proper resources. The data transfer is hidden to the users in order to preserve the easiness of use of the server;
- PIMA(GE)^2 Server acquires data exploiting the suitable tools possibly present on the parallel resources. It is obtained wrapping the efficient I/O operations of the library, described in Section 3.4.3.2. We enable the acquisition of local data as well as remote data, both cases converged in the definition of the I/O module of the library;

- during the execution of client requests, data are always local to the computations. Therefore the client-server interaction during a session has to consider and handle local images, otherwise it leads to an useless movement of large data through the ORB channel;

- the execution of client code has to maintain the efficiency of the native parallel library preserving the granularity of the executions. It has to be obtained without introducing an additional overhead to the server performance.

The first two points of the list enable the reduction of the overhead related to the architecture allowing a better exploitation of the resources, the last two are aimed to achieve an efficient reuse of the legacy code. It is to notice that this strategy does not compromise the easiness of use of PIMA(GE)^2 Server, an example of the client code is presented in Section 5.4.5, and leads to a fairly unvaried efficiency as discussed in Section 5.4.6.

Data Transfer

When the client requires the processing of remote images, it is necessary to move data to computational resources where PIMA(GE)^2 Server is running.

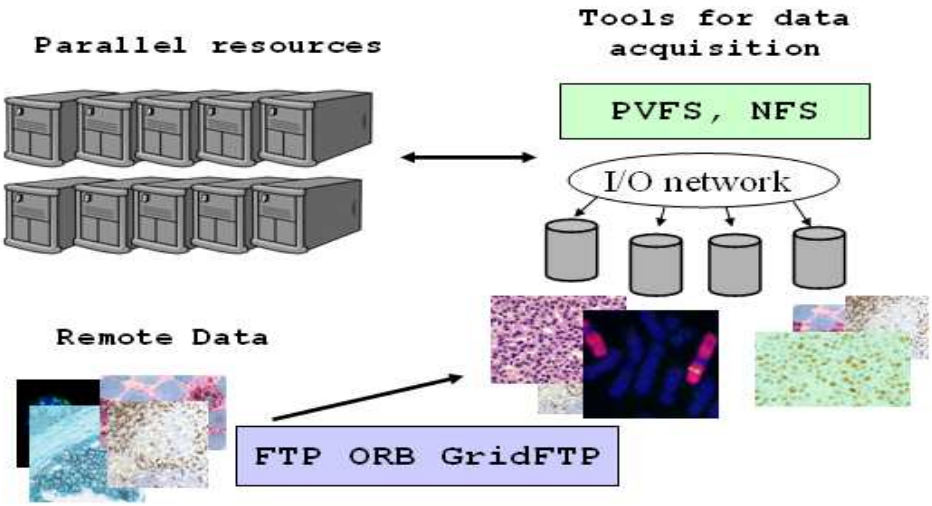


Figure 5.7: *The data transfer can be managed considering efficient protocols, and the data acquisition exploits the suitable tools possibly present on the parallel resources.*

A considerable improvement is brought if data are transferred considering proper tools and protocols; in the CORBA framework, usually the transfer of data exploits the ORB channel. However considering the general scenario of a distributed infrastructure depicted in Figure 5.7, it is possible to consider the use of different protocols to accomplish this

task, as FTP and other. For this reason we experiment the use of protocols that could result more effective in the transfer of large size data.

In particular a comparison between the performance achieved considering the FTP protocols and the ORB channel is detailed in Section 5.4.6. The use of FTP protocols definitely obtains better performance, for this reason we employ it to execute data transfer. To preserve the user-friendliness of the server, we avoid a direct use of such protocols in the client code, and we embedded their exploitation in the I/O operations involving remote data. The I/O functions provided by PIMA(GE)^2 Server are described below, and an outline of the code employing the FTP protocols is given in code box 5.5.

Data Acquisition

Exploiting a parallel machine to execute the computations, we can take advantage of the parallel tools it may provide. As described in Section 3.4.3.2, during the development of the library, we already considered the use of the best I/O subsystems to enable efficient data acquisitions. Also in PIMA(GE)^2 Server it is possible to achieve this goal, and speed-up the data access and distribution. However in the server case, we have to take into account the location of the data with respect to the resources where PIMA(GE)^2 Server is running. Therefore we have enabled the acquisition of remote data as well as local data. This situation is considered in Figure 5.7.

Listing 5.4: *The headers of the I/O operations to acquire data positioned on the server side.*

```
1 // the variable fname contains the filename,
  // the variable IdImgcontains the symbolic
3 // identifier of the images

5 readImageFromFile( fname, IdImg);

7 writeImageToFile( fname, IdImg);
```

If data are local to the resources, their acquisition is performed through the I/O operations of the library. The APIs of local I/O functions are reported in the box code 5.4.

If data are not present on the server site, we move data to such resources in order to exploit the parallel tools possibly present and to improve the client-server interaction. Therefore, the I/O operations to acquire remote data perform two steps: the transfer of data from a specified URL, the acquisition of such data. To accomplish the first task we use the effective protocols for the data transfer discussed in the previous paragraph; to accomplish the second one, we consider the I/O operations of the library since data are now present on the required resources. The APIs of the I/O operations are presented in code box 5.5, where it is also outlined their implementation to show the employment of the FTP protocols. Only the coordinator process is responsible for data transfer.

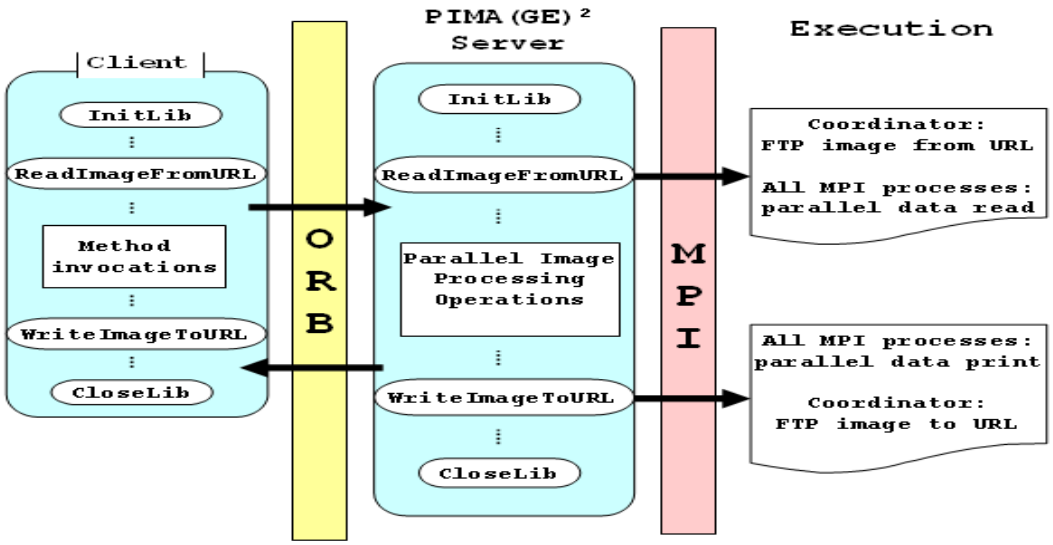


Figure 5.8: *The I/O operations to acquire remote data hide the exploitation of FTP protocols to actually move data in a distributed environment.*

Listing 5.5: *The headers of the I/O operations to acquire remote data and few line of code to outline the exploitation of the FPT protocols.*

```
1
 readImageFromURL( URL, fname, IdImg);
3
 if (myId == coordinator) {
5  char outaux[200];
   sprintf(outaux, ``cd %s; wget %s%s'', DataPath, url,fname);
7  system(outaux);
   } readImageFromFile( fname, IdImg);
9
 writeImageToURL(URL, fname, IdImg);
11
   writeImageToFile(fname, IdImg);
13 if (myId == coordinator) {
   char outaux[200];
15 sprintf(outaux, ``cd %s; wput %s%s'', DataPath, url,fname);
   system(outaux); }
```

The different functionalities developed to enable data acquisition allow the definition an I/O module, that we add the library [56]. Thus the I/O module of PIMA(GE)^2 Lib ensures the speed-up of data access and distribution since it exploits the presence of shared and parallel file systems. In this way it enables a suitable interaction between the primitives of the file system and of MPI-2, see Section 3.4.3.2 for performance evaluations. This behavior

is ensured in both cases, because of the arrangement of the I/O operations provided by PIMA(GE)^2 Server. We developed further I/O functionalities exploiting the Grid, they have been included in the I/O module of PIMA(GE)^2 Lib as well.

Client-Server Interaction

With respect to the client-server interaction, our aim is to minimize the network traffic on the ORB channel in a single session. It leads to a consequent reduction of the overhead introduce in the integration phase. Figure 5.9 depicts an inefficient management of the data movement on the ORB. The input image A is sent to the server twice, and output images C and D are unnecessarily sent back to the client as intermediate output and then to the server once again as input. This represents the common situation in a distributed infrastructure, we have to introduce a better strategy to avoid this redundancy in the communications.

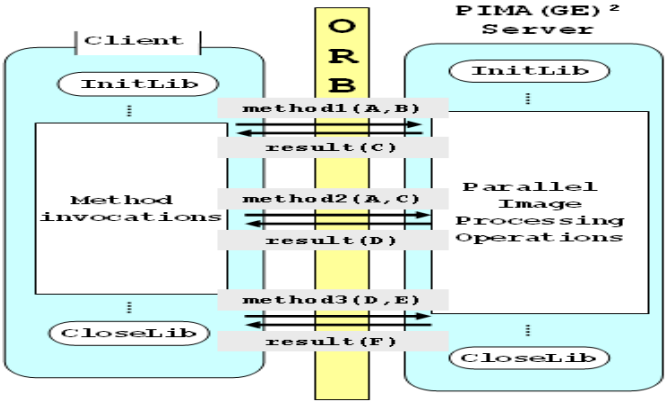


Figure 5.9: *Without an actual policy to optimize the client server interaction a great amount of data is uselessly moved in a distributed environment.*

Since the execution of a single client-server session is entirely performed on a fixed resource with the same degree of parallelism, data can be distributed once and kept on the parallel nodes till the end of the computation. Therefore we can refer to the stored data using a symbolic identifier and avoid useless movement on the ORB channel. Each identifier corresponds to a unique image during a single client-server session. For each images elaborated or handled in a code, the client has to perform a standard declaration of an integer variable. The actual link with the corresponding image is performed automatically by the server in the acquisition or creation operations. The client is not allowed to assign a value to the symbolic identifiers. When the client-server session ends, the symbolic identifiers are released. An example of client code, and further details are given in Section 5.4.5.

The use of symbolic indefinites leads to the situation depicted in Figure 5.10, that introduces a consistent reduction of network traffic with respect to the situation considered

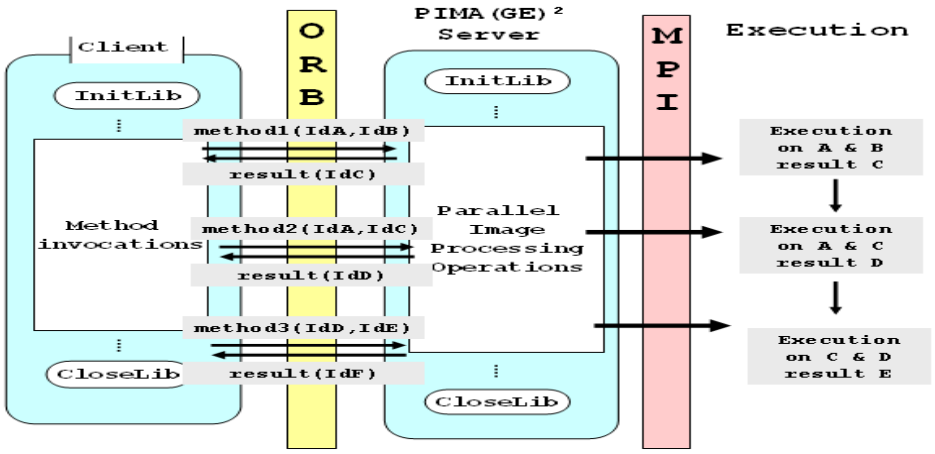


Figure 5.10: *Considering a policy to avoid useless data movement during the client server interaction it is possible to reduce the overhead of the computation.*

in Figure 5.9. In this case, only symbolic identifiers are exchanged between clients and the server; it actually minimizes the utilization of the ORB channel during a session, as outlined in tests presented in 5.4.6.

Client Code Execution

With respect to data processing, our aim is to preserve the granularity of the computation in order to enable an effective reuse of the library code. In the adopted general scenario, to achieve this goal we combine the use of symbolic identifiers, and the exploitation of the coordinator process described in Section 5.4.3.

It is possible to process data in parallel. We already assume that it is not actually necessary to distribute the images for each invocation of methods, otherwise it results in an useless data movement. Thus data can be distributed once and kept on the parallel nodes till the end of the computation. Images are partitioned among the parallel processes during the operations of acquisition or creation. In this phase we link data to the symbolic identifiers.

For each parallel process involved in the computation, the symbolic identifier points to its partial data, i.e. the part of the global image that the single process is responsible to elaborate following the user requests. Figure 5.11 depicts this situation considering 4 MPI parallel processes. This situation acts as the optimized parallelization policy applied in PIMA(GE)^2 Lib, therefore it is possible to enable efficient code reuse through a proper wrapper. Through the coordinator process, the parallel MPI processes receive the information and execute the operations considering their partial data, that are processed following the parallelization policy applied in the native library. In this way we enable a direct interaction between the client and the library operations. This interaction is kept alive until the invocation of `CloseLib`. The coordinator gathers the image when specifics

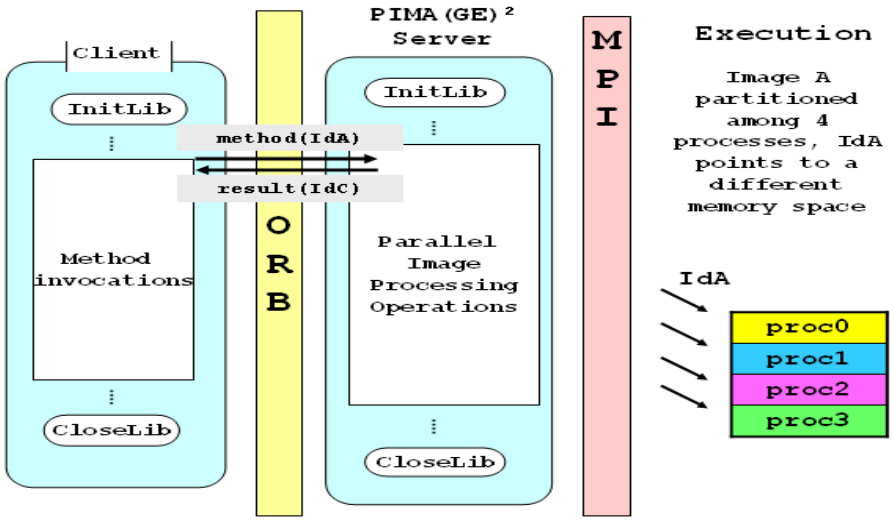


Figure 5.11: *In this example the image A is partitioned among four processes, the symbolic identifier could point to a different memory location without violating the computation.*

operations are required, e.g. to print an image. Tests to validate this strategy give positive results, as presented in Section 5.4.6.

5.4.5 An example of client code

In this Section we provide a simple example of client code to show the easiness in the development of image processing applications using PIMA(GE)^2 Server. We present a simple application elaborating the input image `origImg` and giving back the output image `resImg`; it is presented code box 5.6.

As it is possible to see, the Client code employs the methods provided by PIMA(GE)^2 Server in a standard way. In fact the policy applied to allow MPI-CORBA compatibility is totally hidden to the Client and managed by PIMA(GE)^2 Server, as well ad the parallelism of the computation.

The first part of the code, lines 3-9, is aimed to start-up the ORB and the IOR, and to obtain a reference to a PIMA(GE)^2 Server instance. They represent standard mandatory steps that each CORBA client has to perform to exploit CORBA objects. After that the PIMA(GE)^2 Server methods are requested and the image processing computation starts.

Lines 14 e 32 report the mandatory steps of initialization, `InitLib()`, and closure, `CloseLib()`, of the library. They are used to enable the the start-up of the parallel environment and coexistence of the MPI environment in the CORBA framework. The degree parallelism is

defined the server.

Listing 5.6: *An example of client code. The client requires a rotation and a reflection of the input image OrigImg, the result is the output image resImg.*

```
1 int main(int argc, char* argv[]) {
2
  CORBA::ORB_var orb = CORBA::ORB_init (argc, argv,"");
4 std::string ior; std::ifstream is(argv[1]);
  std::getline(is, ior);
6 CORBA::Object_var obj = orb->string_to_object(ior.c_str());

8 //get the pimage object reference: module LIB object Operation
  LIB::Operation_var pimage = LIB::Operation::_narrow(obj.in());
10
  //declaration of the symbolic identifiers of images
12 CORBA::Long origImg, resImg;

14 pimage->InitLib();

16 //implicit association of the integer origImg to the image
  //remotely stored in the file "myimg" through the FTP service
18 pimage->readImageFromURL("http://www.disi.unige.it/GaliziaA/",
  "myimg",origImg);
20 pimage->getdimension(origImg,imWidth, imHeight, imDepth);

22 //implicit association of the integer resImg to a new image
  pimage->createImage(imWidth, imHeight,imDepth, resImg);
24
  pimage->rotImg(IdImg, resImg, 60, LINEAR, 0, p);
26 pimage->reflectImg(resImg,doX,doY);

28 //we print locally the image
  pimage->writeImageToFile(resImg,"myresult");
30 pimage->deleteImage(origImg);
  pimage->deleteImage(resImg);
32 pimage->CloseLib();

34 orb->destroy();

36 return 0; }
```

For each image elaborated or handled in a client code, a symbolic identifier has to be used. It corresponds to a standard CORBA declaration of an integer variable, line 12. `origImg, resImg` identify the unique correspondent images during all the client-server session minimizing the use of the ORB channel. Their actual connection with the corresponding image is performed automatically by the server during the acquisition or creation operations. In the example code they are related to the corresponding images through a

read operation, and a create operation:
`readImageFromURL(URL,"myimg",origImg)`,
`pimage->createImage_C(imWidth, imHeight,imDepth, resImg)`.
They corresponds to an implicit association of the identifier to its image; from now on, the client considers the identifiers to elaborate the corresponding images. Once the client image processing is finished, `origImg, resImg` have to be released (lines 30 and 31). It is also necessary the destroy the instance of the ORB, line 34.

5.4.6 Experimental results

We carried out tests aimed to evaluate the effectiveness of the PIMA(GE)^2 Lib integration, through the analysis of the overhead related to the exploitation of a distributed infrastructure. Therefore we consider two aspects:

- the evaluation of the possible benefits obtained exploiting the FTP protocols to transfer data on the computational resources. In this direction we compare the time spent to transfer the same amount of data through CORBA services using in turn FTP protocols and the ORB channel;
- the use of symbolic identifiers, i.e. the integer variables to identify the actual images, during the client-server interaction for a sequence of image operations. In this case, we consider a complex application to perform the tests is aimed to the detection of linear structure in a TMA image, and we want to evaluate the variation of the speed-up value considering the native library and the server.

To execute the tests we consider as parallel execution environment a Linux cluster with eight nodes interconnected by a Gigabit switched Ethernet, each node is equipped with a 2.66 GHz Pentium processor, 512 Mbytes of RAM and two EIDE disks interface in RAID 0. The client is in the same local area network to limit other possible sources of overheads. In fact we avoid to use a "true" remote client, since we would like to measure the overhead due to PIMA(GE)^2 Server infrastructure, without the possible unpredictable behavior of an actual remote interconnection. From the data transfer point of view, we developed two CORBA services to transfer data from the site specified by the client to the computational resources where PIMA(GE)^2 Server runs. They consider the use of FTP protocols and the ORB channel to transfer the same amount of data between two fixed sites. We prove that definitely better performance are obtained exploiting FTP protocols; the execution time using the ORB channel is about twice of the one obtained using the ftp protocol. This behavior is independent from the data size, according to the results presented in Figure 5.12. As already explained, the service exploiting FTP protocols have been integrated in the I/O operations, and its use is completely hidden to the users.

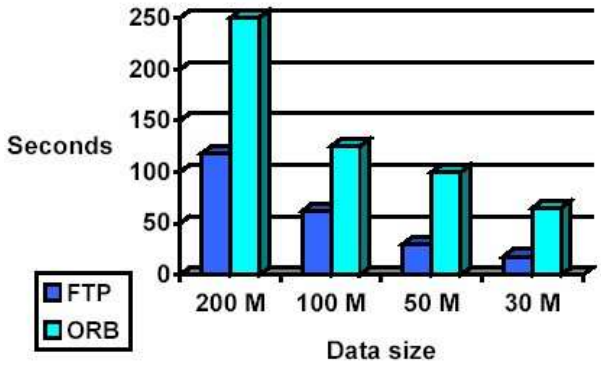


Figure 5.12: *Graphics reporting the comparison between the data transfer services implemented considering the FTP protocols and the ORB channel.*

From the application point of view, we considered three cases:

1. the application is developed using PIMA(GE)^2 Lib and is executed on a local parallel resource (i.e. without the integration in the server);
2. the same application has been added to the PIMA(GE)^2 Server operations, it represents a single complex method executed on the same resources of the previous case;
3. the application is implemented as a sequence of methods of PIMA(GE)^2 Server executed on the same resources of the previous cases.

We evaluate the variation of the speed-up values in these conditions, they are depicted in Figure 5.13.

The first test is necessary in order to define a term of comparison for PIMA(GE)^2 Server. In the second test, the interaction between the client and the server is very coarse grained and we have a minimum number of communications on the ORB; there is only one method invocation besides the mandatory invocations of `InitLib` and `CloseLib`. Thus we are actually measuring the overhead due to the CORBA start-up. As expected, this start-up is a fixed time (about 1-2 seconds in our experiments) that does not depend on the number of parallel MPI processes. The third test is aimed to measure performance when a more consistent use of the ORB communication channel and a growing interaction between the client and PIMA(GE)^2 Server. In fact in this case the application calls about 2166 server methods. It leads to performance degradation, but applying the symbolic identifiers strategy the overhead is quite limited especially for compute intensive applications. In the tests we obtain a fairly unvaried speed-up performance. Comparing the speed up of the application implemented with the library functions and one implemented with the

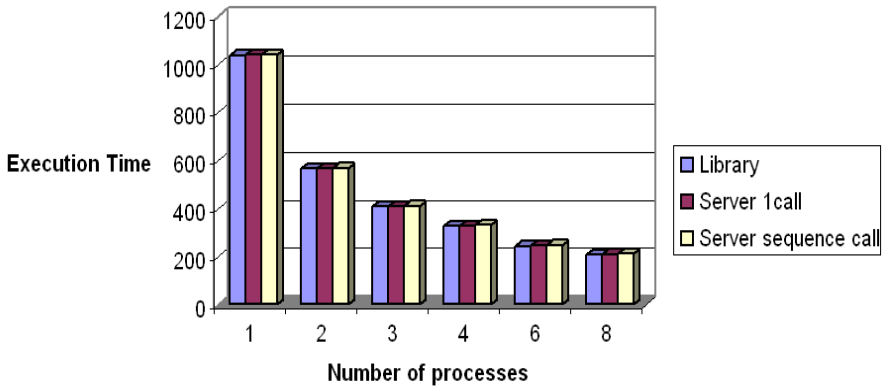


Figure 5.13: *Graphics reporting the comparison among the speed-up of an edge detection developed through the native library operations, and the* PIMA(GE)^2 *Server. We considered in turn the edge detection as a complex* PIMA(GE)^2 *Server method, and as a sequence of simpler methods.*

PIMA(GE)^2 Server methods, the variation is around 1.3%; experimental results are plotted in Figure 5.13.

Chapter 6

Exploiting the Grid

6.1 Motivations and goals

In this Chapter, we present how to enable the use of a parallel image processing library in a Grid environment. We already introduced the Grid architecture in Chapter 1, and we analyzed the general scenario for distributed/Grid processing in Chapter 4, where we also made several hypotheses on the computations. In the Grid environment we assume the existence of a middleware that implements a distributed virtual infrastructure; more specifically we consider two different situations, in one case we adopted Globus [51], and in the other gLite [60]. It is possible to consider different situations to enable the use of a parallel library, and it is important to obtain a good exploitation of the available dynamic computing resources. In particular, data could be stored on a remote host as well as the output of the computation.

Starting from these hypotheses, we consider three different aspects:

- we develop Grid services to enable the integration of PIMA(GE)^2 Lib in a Computational Grid, and an efficient exploitation of Grid resources;
- we enable the transparent use in the library code of Grid protocols aimed to ensure secure data movement and acquisition;
- we develop a framework that provides library functionalities in order to build vertical Grid applications.

To achieve the first goal we employ basic Grid services provided by the Globus middleware version 4, while in the other cases we consider EGEE [48] and the gLite middleware. In the second work, we employ services already developed in EGEE, in particular the GFAL API

provided by gLite to improve the features of the library. In fact in this way we ensure a secure data management, that represents a key issue in the processing of biomedical data. In the third case, the application domain of the framework is the TMA images analysis [86]. In Section 3.5, we already detailed the development of specialized functions aimed to accomplish this task, in this Chapter we describe how we enable their use on the Grid.

6.2 Related works

In order to integrate existing software tools in a Service Oriented Architecture, there are two possible solutions: the re-engineering and re-development of the code, or the encapsulation of the code with a proper technology [124]. The second solution is clearly the most interesting when complex applications have to be integrated in the Grid environment, and it is the subject of many research efforts.

Specific architectures have been developed to enable the execution of legacy code applications on the Grid. Their aim is to allow an automatic deployment of legacy applications as Grid services. An example of semi-automatic tool has been developed at the Cardiff University [69]. Their approach is based on the semi-automatic conversion of C programs into Java using Java Native Interface (JNI). After wrapping the native C application with the Java C Automatic Wrapper (JACAW), the MEdiation of Data and Legacy Code Interface tool (MEDLI) is used for data mapping to make the code available as part of a Grid workflow using Triana [134].

Another example is the EasyGrid framework [17], which considers specifically the execution of MPI legacy programs across clusters of distributed computing resources. The framework is based on the idea of developing system-aware applications automatically generated from the user's MPI source code. It considers mechanisms to redirect messages through the code wrapping of MPI functions.

Also GEMLCA [37] provides a framework to deploy existing applications as a Grid service. It assumes that the legacy application runs in its native environment on a Compute Server and requires only a user-level understanding of the legacy application. The deployment defines the execution environment, and the legacy code is considered as a binary file. Also MPI applications can be deployed. In order to provide a Web interface (portal) to access the GEMLCA functionalities, GEMLCA was integrated with the workflow-oriented P-GRADE Grid portal [121] extending its functionalities.

The main drawback of these architectures is that they allow the integration of single applications, while we need to efficiently integrate a parallel library and old and new code developed using it.

The GrADS project [65] is probably the most remarkable example of integration of parallel

libraries. In particular, the widespread ScaLAPACK [16] and PETSc [13] libraries were considered and a set of experiments towards the porting of specific functions were made. Each legacy function is wrapped in the GrADS framework through the definition of a specular GrADS service where the native function is called [100]. Their approach is not completely suitable in our case because each library function is integrated as stand alone operation. This approach may worsen the performance achievable with the original library.

If we consider Grid systems aimed to image analysis, works concerning different application domains are presented, as for example remote sensing [102, 118], and astronomical imaging system [144, 9], but most of the projects and tools are designed for biomedical images processing [59, 3, 79]. Indeed the Grid represents a suitable infrastructure to accomplish this task [78], since it provides storage facilities and high computational power under a security infrastructure. Grid is often exploited to ensure restricted data access in bioinformatics projects.

The cancer Biomedical Informatics Grid (caBIG^{TM}) project [20] connects scientists and organizations to combine strengths and expertises in an open environment. It relies on the caGRID Grid infrastructure [21], that is aimed to bring together data from many and different data sources, and to enable data analysis. Issues related to privacy and security are addressed in the Data Sharing and Intellectual Capital Workspace (DSIC WS). They faced security aspects following the GSI method, and developed an architecture very similar to the EGEE system.

The Grid Enabled Microarray Experiment Profile Search (GEMEPS) project [120] aims to develop a Grid infrastructure for discovery, access, integration and analysis of microarray data sets. The aim of the project is to highlight the efficacy of Grid technologies for dealing with heterogeneous security infrastructures and federated microarray data environments, and to provide researchers to securely and selectively share sensitive, pre-publication microarray data. They simplify the Globus authorization and authentication, users can access resources by using their own home security domain to perform the authentication. It is made through a portal; once logged-in a user can exploit the portlets depending on its level of privilege and without repeating the authentication process.

A similar approach has been proposed in the Biomedical Research Informatics Delivered by Grid Enables Services (BRIDGES) [119], which deals with the federation and the processing of distributed Cardiovascular Functional Genomics data exploiting Grid services. They consider the use of a portal to check the resource access and to simplify the authentication and authorization phases. They exploit PERMIS to provide a fine grained security (authorization). The data management is implemented with different security levels. Actually data integrated in BRIDGES can be public domain or restricted for usage only by specific VO partners.

6.3 The EGEE Grid Environment

EGEE is a wide area Grid infrastructure for scientific applications relying on the Globus Toolkit. EGEE classifies the Grid resources in Computing Elements (CE) and Storage Elements (SE); thus the EGEE infrastructure is a network of several CEs, that are the gateways for the Worker Nodes (WN) on which jobs are performed, and a number of SE, on which data are stored. To exploit the resources the users must be members of a VO.

Through the User Interface (UI) it is possible to submit jobs, monitor their status and retrieve the outputs if jobs are terminated successfully or resubmit them in case of failure. The distribution of the computational load on the CE is performed by the Resource Broker (RB). Jobs are submitted from a UI using scripts written in the Job Description Language (JDL), which specify the necessary directives to perform the task.

The Grid infrastructure implements a set of tools in order to manage data in a distributed way. These components provide accessibility to the place where files are physically located and to the LCG (LHC Computing Grid - Large Hadron Collider -) File Catalog (LFC) system used to associate a physical identifier to the Logical File Name (LFN). The LFC contains Globally Unique Identifiers (GUIDs), i.e. identifiers for mapping logical and physical files. Both, the GUID and the correspondent LFN can be used to access data. The LCG system provides the functionality to the data management. They enable a transparent interaction with file catalogs. Files are visible to the users belonging to the same VO, but the owner can manage permissions, and avoid the access by non-authorized users.

We exploit EGEE during the Thesis, in particular we consider the use of the Grid File Access Library, i.e. GFAL, and of the official metadata service for EGEE, ARDA Metadata Grid Application (ARDA Architectural Roadmap toward Distributed Analysis) i.e. AMGA. Their features are detailed in Section 6.5 and 6.6. In Chapter 4, we outlined that an efficient data management could be obtained through a separation among data transfer, access and processing. It is to notice that in EGEE the separation between data access and processing is mandatory, taking into account the classification of Grid resources as CEs and SEs.

6.4 Developing Grid services to enable library code executions

In this context we consider the exploitation of the Grid to enable the use of PIMA(GE)^2 Lib and the efficient execution of the application developed with its operations. Our idea is to develop a system where the user has to provide the source code of the application

developed using the library and the system is responsible of its efficient execution. We call this system PIMA(GE)^2 Grid, our aim is to evaluate the complexity and the cost of the integration of the library in a Computational Grid.

To achieve these goals we need to develop several Grid-oriented functionalities to enable an efficient use of the native library. The major topic is to develop a service to accomplish the resource selection properly, i.e. to select the suitable resource to obtain efficient execution. In this phase, a key topic is to allow the use of the scheduling policy defined in PIMA(GE)^2 Lib when heterogeneous resources are exploited. The policy has been described in Section 3.4.1.1, our aim is to take into account the heterogeneous computational power to execute the application on Grid resources. Further services have been developed, they are aimed to provide basic functionalities such as: the staging of I/O file and of the source code, i.e. transfer of these files, the compilation of source code of the application, the execution of the corresponding parallel job, the clean up of files necessary for the computation. To achieve these goals we consider Grid Services implemented through the Globus Toolkit version 4. We implement a prototype of the system and we test it exploiting a local Grid in Genoa.

From the user point of view, PIMA(GE)^2 Grid acts as an intermediate layer between Grid resources and parallel image processing applications. We design it considering the abstraction model proposed in [4, 111], the aim is to provide an application development framework that shields the Grid users from the technological complexities of the Grid implementation and of the execution of parallel applications on the Grid. PIMA(GE)^2 Grid virtualizes resources and services, and avoids the technical and semantic complexities of the middleware. Actually, Grid services aimed to image processing and to the exploitation of the architecture have been deployed in advance, and through PIMA(GE)^2 Grid, they are coordinated and provided to the users. In this Section we provide a description of the work.

6.4.1 Using Grid services to enable the use of PIMA(GE)^2 Lib in a Grid

To develop application through PIMA(GE)^2 Lib and execute them on the Grid, it is necessary to perform several steps; on the base of the remarks made in Chapter 4, we consider separately data movement, acquisition and processing. These steps can be summarized in the list below:

1. discovery the resources available on the Grid,
2. select the architecture to execute the computation;
3. compile the code provided by the users considering the selected architecture;

4. select the actual computational resources to execute the computation;
5. transfer the input data and the executable on the selected resource(s);
6. launch and monitor the execution;
7. transfer the output data on the machines specified by the users;
8. remove all the files from the selected resource(s).

Our aim is to leave to the user only the development of the application using PIMA(GE)^2 Lib, while the management of the other steps could be completely delegated to automatic tools and services. The major issue is to limit the unavoidable overhead due to the Grid environment, in particular considering the need to move the data among computational resources and the users using low bandwidth links.

Starting from these remarks, we exploit three Grid services to perform the steps previously presented:

- **PIMA(GE)^2 GS**, it is the actual image processing service, since its purpose is to copy and execute the parallel image processing applications developed using PIMA(GE)^2 Lib. It is also responsible for the compilation and interacts effectively with the other Grid service to enable parallel image processing computations;
- **GEDA**, **Grid Explorer for Distributed Applications** [26], a Grid Service designed to provide a simplified access to Grid Resources. GEDA is a general purpose resource broker service that can be used to allow a fine grain resource selection for any kind of Grid service. In this work we extend its functionalities in order to integrate the scheduling policy applied in the library; it is detailed in the remaining of the Section;
- the **GridFTP** file transfer service, provided by the middleware GT4.

They compose PIMA(GE)^2 Grid framework, their specific aims are detailed in the remaining of the Section. PIMA(GE)^2 Grid provides its services to the users through a Java client application, its description and the general behavior of the framework are described below.

PIMA(GE)^2 Grid actually implements the separation of the three topics in the data management to enable an effective code reuse and efficient executions. Indeed the granularity of the application developed through the library is preserved, and considerations on the applications performance coherently reflect the efficient data management in the Grid environment. Moreover through GEDA we are able to select the most suitable resources available on the Grid, and to apply the scheduling policy of the library to execute the parallel computations.

6.4.2 Inside PIMA(GE)^2 Grid

The schema presented in Figure 6.1 shows the interaction between the components of PIMA(GE)^2 Grid, the middleware and users. There are four groups of logical functionalities that the user may access by means of the PIMA(GE)^2 Grid interface:

- Compilation represented by arrows 1 and 2;
- Resource Selection represented by arrows 3 and 4;
- File Transfer represented by arrow 5;
- Execution represented by arrows 6, 7 and 8.

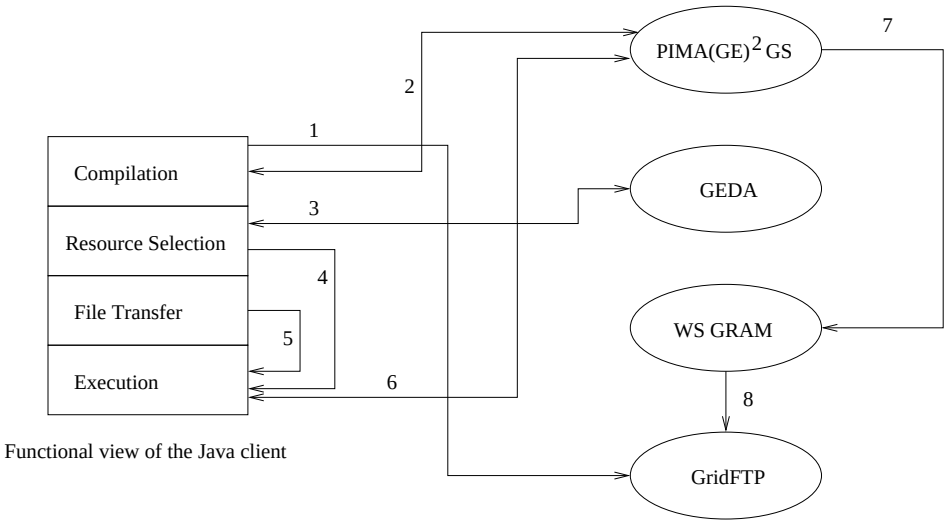


Figure 6.1: *Interaction among the various entities involved in the execution of a user application using* PIMA(GE)^2 *Grid. Numbers represent the order in which they take place.*

The **Compilation part** consists in the transfer of the C source code of the program that the user wants to execute, and in its compilation by PIMA(GE)^2 GS using PIMA(GE)^2 Lib and MPI-CH 2. The resulting executable is stored on the disk, and will be transferred on the resources selected for the execution. Thus, the compilation part exploits the **File Transfer**.

The **Resource Selection** is provided by Grid Explorer for Distributed Applications (GEDA) a Grid service aimed to resource selection to aid the execution of parallel jobs [26].

Considering possible parameters specified by the user through the interface, see Figure 6.3, GEDA monitors the available resources on the Grid, and provides a description of each resource to the users. GEDA sorts the available resources ordered on the workload, the first element corresponds to the resources selected to execute the current parallel application.

When a resource is selected, it is necessary to define the number of process to spawn for the execution. If the resource is an homogeneous machine, by default the number of processes that will be launched is equal to the number of processors, but it is possible to modify this value. Considering that a computational resource in the Globus middleware is represented by the URI of its WS GRAM, for an homogeneous machine the result of the selection step is a pair specifying the URI and the number of processes to be spawned, i.e. *(URI, parallelism degree)*.

Instead if the resource is an heterogeneous parallel machine, GEDA applies the scheduling policy of the library in order to enable a suitable exploitation of such resource. To achieve this goal, we integrate the scheduling policy in this way, when GEDA selects an heterogeneous parallel machine, the benchmarking function is run in order to derive the relative velocity V_i of each nodes. Once known these values, GEDA executes the scheduling algorithm in order to calculate the number of processes to spawn on each heterogeneous node. In this case, if the resources is composed of N nodes, the result of the selection step is a (N + 2)-uple that specifies the URI of the WS GRAM of the selected resource, the global number of processes, and the number of processes to spawn on each single node, i.e. *(URI, global parallelism degree, . . . , parallelism degree on the i-th node,. . .)*.

The **File Transfer part** is represented by the specification of the URIs of the I/O files of the program as well as of the source code of the applications. These files will be transferred to/from each computational resource using the GridFTP protocol. These transfers are made in conjunction with the execution of the application, managed by the **Execution part**, or in conjunction with the **Compilation part**.

The **Execution part** is performed by PIMA(GE)^2 GS that produces an RSL2 file containing all the necessary parameters. To actually run the job, PIMA(GE)^2 GS collects the information of the previous parts, and considers the resource selected by GEDA and its degree of parallelism. If the job is submitted on a homogeneous machine the parallel processes are equally subdivided on the nodes, otherwise the execution takes into account the (N + 2)-uple provided by GEDA. The resulting file is then submitted to the WS GRAM of the computational resource selected to run the current application.

6.4.3 PIMA(GE)^2 Grid User Interface and Functional Description

The user interacts with PIMA(GE)^2 Grid using the graphic interface of a Java client application, which is presented in Figure 6.2. We suppose that the user already developed

the application to execute on the Grid through the native version of the library. The user is responsible to produce a correct code, i.e. without errors during the compilation phase.

The first parameter to be provided is the Uniform Resource Identifier (URI) of PIMA(GE)^2 GS. Then the user has to specify the location of the C source code of the application, and the kind of platform to use. The code file is uploaded using GridFTP, then it is compiled and linked with the library. Presently, we consider only the use of 32-bit Linux architectures.

The next step is the selection of the computational resources through GEDA. We provide in the PIMA(GE)^2 Grid Java interface also a client for GEDA, see Figure 6.3, to specify the characteristics of the resources and visualize the corresponding available one. In particular in the current version of the system the research is limited by defaults to only 32-bit Linux architectures.

Last step to enable the execution of the application is the indication of the URIs of input and output files through the interface of PIMA(GE)^2 GS. In the present version we transfer the I/O data using the GridFTP protocol. The actual data acquisition is performed employing the efficient I/O operations provided by the native library.

When all the parameters are specified the execution of the application on the selected resources starts. During the execution a feedback is provided by PIMA(GE)^2 GS in order to monitor the status of the of the job. We exploit the `GramJob` class to accomplish this task.

It is also possible to require the output of the program as a further file, for example in order to analyze statistical information, as the times of the different parts of the application, or to get error messages.

6.4.4 PIMA(GE)^2 Grid evaluation

PIMA(GE)^2 Grid experience provides us indications about the effort necessary to make PIMA(GE)^2 Lib available in the Grid.

It is now clear that a basic Grid enabled version of the library can be obtained with a limited effort, on the currently available middleware. The separation among data transfer, acquisition and processing turn out to be once again a good strategy to limit overhead due to the complexity of the environment.

A specific experimentation should be done about the efficiency of GEDA as a tool to make the more convient selection of computational resources. However this implies experimentations on largest Grid, and it is out of the scope of this Thesis. We may claim that we checked mechanisms but other effort will be necessary to verify resource selection strategy. This is left for future research.

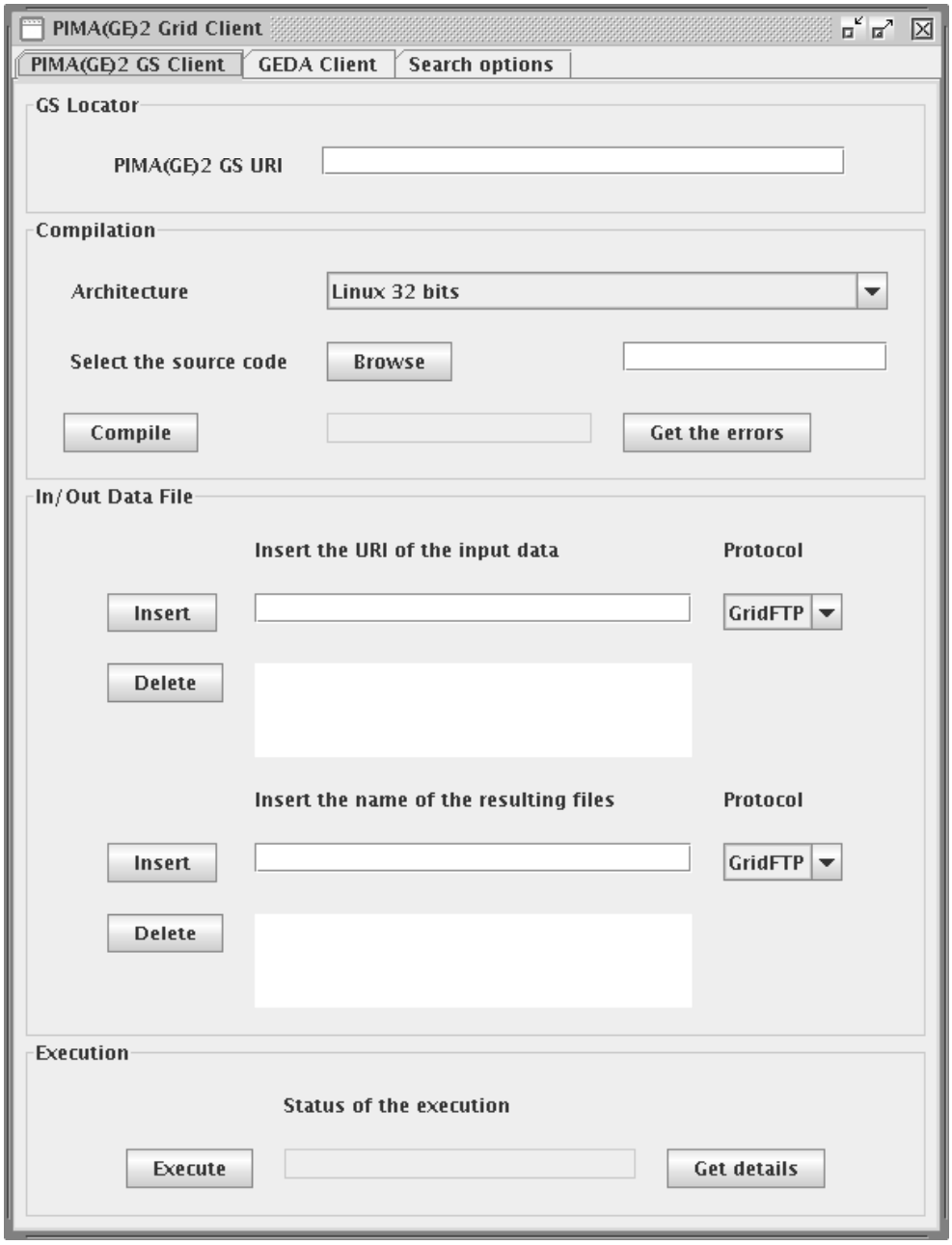


Figure 6.2: *The interface of the java client that permits to interact with* PIMA(GE)^2 *Grid.*

6.5 Exploiting Grid protocols to improve the library features

Our aim is to experiment Grid protocols of gLite to improve privacy preservation of sensitive data, and thus to extend the features of PIMA(GE)^2 Lib. Our idea is to develop

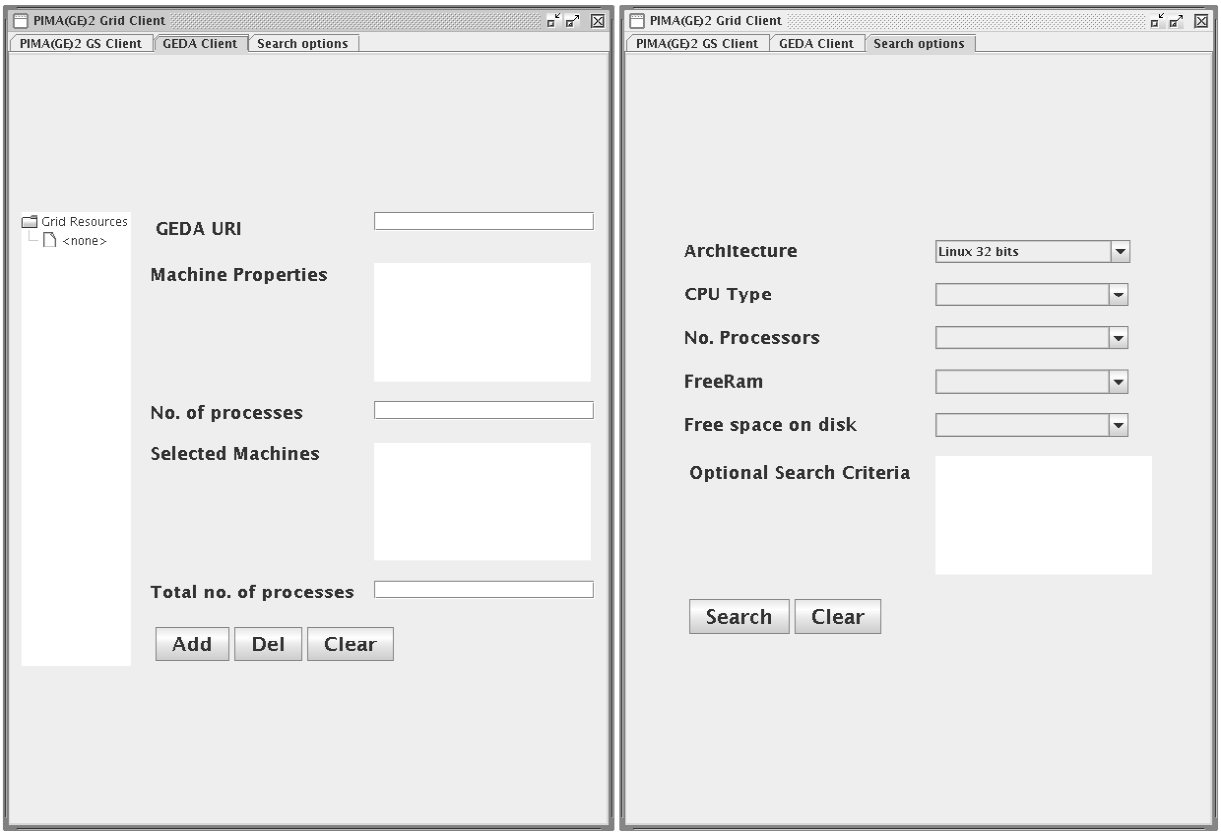


Figure 6.3: *The interface of the java client that permits to interact with GEDA.*

a set of specialized I/O functions to enable secure remote data access; these operations have to transparently verify the authorization policy in the data access possibly defined by the VO. In this way we comply with the policy imposed by the owners of data without compromising the easiness of use of PIMA(GE)^2 Lib.

To achieve this goal we exploit specific functionalities provided by gLite, in particular GFAL, Grid File Access Library. GFAL is a library aimed to file management and provides API for different programming languages. It implements functionalities for the explicit interaction with the file stored through the LCG File Catalogue, ensuring the abstraction from technology of specific implementations, and transparency during the interaction with file catalogs and storage interfaces when needed. In particular, GFAL functionalities enable the access of file located in SEs, and their processing without copying data on the WNs. Thus, avoiding multiple physical copies of data, also any possible data intrusion is unlikely. In fact data are not present on a machine different form the SE where they have been allocated under a strict access policy. The separation among data transfer, access and processing in this case is not respected. In fact it is to notice that considering the use of

GFAL, data transfer and acquisition take place at the same time, but it also ensure an effective data management in a distributed environment.

We transparently integrate the use of GFAL in the library code to develop I/O operations on remote data. They enable secure remote data access and processing without copy data from their original position. The architectural domain is therefore EGEE and gLite. The specialized set of I/O operations have been added to the I/O module of the library, and have been tested considering Computing Elements and Storage Elements at different research institution in Genoa and Milan. During the remaining of the Section we detail this work, and we provide an analysis of the performance.

6.5.1 Exploiting further protocols in EGEE to improve the features of PIMA(GE)^2 Lib

In the image processing community, a topic of interest is represented by privacy preservation and secure handling of sensitive data, such as biomedical images. In this context images are related to confidential information corresponding to real patients, and it is necessary to ensure secure access and privacy preservation. Thus only a restricted data access could be allowed, through an authorization policy and the user identification. Moreover in most cases, to copy patient data in places different from the original database is forbidden by the owner institutions.

The Grid enables data sharing and computations under a solid security infrastructure, thus we consider its use to adapt the library code to analyze distributed images complying with policy imposed by the owner institutions. Actually our efforts are oriented to the observance of the already applied policy rather than the definition of further policy to restricted data access.

Most of the projects presented in the last part of Section 6.2 ensures restricted data access through the VO authorization policy and the Grid Security Infrastructure (GSI). In this work we use the EGEE security infrastructure for the authorization and the authentication processes. However we add a further level of security in the processing of sensitive data through GFAL.

Combining the use of the Grid Secure Infrastructure, an appropriate VO sharing policy, and the GFAL API restricted data access can be ensured. In fact, data are still under a restricted access policy, and avoiding data movement and copy among geographically distributed resources, any possible data intrusion is avoided.

We employ the GFAL read/write functions for remote data, in this way we are able to acquire data directly form the SE where images have been stored and accessed under a strict control. We hide the use of the GFAL API properly integrating the GFAL functions

calls in the implementation of the I/O operations of PIMA(GE)^2 Lib. We modify the I/O operation API to add another possibility in the selection of the protocols to employ in the I/O management, and in this way the use of the GFAL API becomes transparent. The interface to read remote data becomes:

Listing 6.1: *API of the operation to read remote data.*

```
1 int readImageFromURL(const char * URL, const char * fname,
2         IMAGE * origImg, enum PROTOCOL);
```

where the variable `URL` contains the path of the file, the variable `fname` the filename, the variable `IMAGE` the pointer to the data structure IMAGE where data are store, the variable `PROTOCOL` specifies which the kind of I/O to apply. The possible values of `PROTOCOL` are: `FTP, GFAL`. The FTP case as been described in Chapter 5.

When the GFAL is selected, PIMA(GE)^2 Lib exploits the functionalities as presented in box code 6.1 Considering data stored in the variable `buffer`, to read remote data sorted in `path` is possible with the following instructions:

Listing 6.2: *Pseudocode of the operation to read remote data; the use of GFAL is outlined.*

```
1 %\begin{verbatim}
2     if ((fd = gfal_open(fname,  O_WRONLY | O_CREAT, 0644)) < 0)
      {
4         printf("Gfal file %s open failed (%s)\n",fname,strerror(errno));
          exit(-1);
6     }

8         ....

10    gfal_read(fd, buffer, n);

12        ....

14    gfal_close(fd);
```

The level of easiness of the PIMA(GE)^2 Lib functions in the actual application development is not compromised. Of course the GFAL API do not support the parallel access to the remote, therefore it is not possible to apply the parallel I/O of MPI 2. It is necessary to consider a master-slave approach to distribute data; thus through the GFAL functions a single process accesses and acquires data in main memory, and after scatters data among the MPI processes.

6.5.2 How to perform computations

We develop the edge detection using the library functions and specifying the GFAL protocol to access data. In order to exploit GFAL during I/O operation it is necessary to execute the computation on the Grid, in accordance with a standard EGEE submission.

Thus the user has to identify as member of the VO; through the UI, the application has been submitted using the JDL script, that describes how the job has to be computed on the WN. This step has to be performed by the user; in TMAinspect, the Grid portal described in the next Section, the job submission is automatically performed by the system, and the user is not involved in this phase.

The test considers 10 images stored in a set of SEs, acquiring raw data and the metadata not related with patient information. They are acquired one at a time using the I/O functions employing GFAL, processed on a CE, in the specific case a remote workstation, and then the output image is produced with respect to the defined privacy policy.

Tests gave positive results also in the execution time, in order to demonstrate the effectiveness of this approach, the same application has been executed using different protocols in the data acquisition. In particular, the attention was focused in the time that is spent on the data acquisition.

The aim of the tests is to compare different situations for distributed image processing. In all the cases there are: a client that drives the execution, a remote data storage and a remote computing node. The point is to evaluate performance and security in data transfer between the storage and the computing element. Data were stored on SE in Milan, and the remote computing node is in Genoa. In a first test the three elements interact using FTP protocol outside of the Grid infrastructure. In the other two cases the elements interact within EGEE Grid infrastructure, in one test using LCG, in the other GFAL to transfer data. A schema of the obtained results is given in Table 6.1.

The most efficient operation is the FTP data transfer, requiring 12 seconds in the management of a 10 MB image. It is an almost evident result, because this protocols is not involved in a complex Grid infrastructure and does not suffer from the overhead related to it. Furthermore this service does not apply any strategy to obtain a secure data transfer, in fact data are sent in clear text, and can be intercepted by everyone. This step also improves the service performance, however its use in the bioimaging community has to be discouraged.

The similar operation provided by the EGEE infrastructure, i.e. LCG-copy, requires about one minute (65 seconds) to manage the same image. In fact in this case the data transfer suffers from the overhead related to the Grid protocol, but data are transferred in a secure way. This service should be used instead of the previous service.

Protocol	Transfer Rate	Security
FTP	853 KBps	None
LCG-copy	365 KBps	VO
GFAL	157 KBps	VO without multiple copy

Table 6.1: *A schema of the comparison among different protocols to transfer and acquire data.*

The data acquisition through the use of GFAL requires 28 seconds. This service also suffers from the Grid overhead, however its performance are really better than the LCG-copy one, and it still provides a secure data acquisition. Actually, since the data are not physically transferred, it represents the most secure way to access data.

6.6 Developing Grid vertical applications for TMA analysis

In this work we enable the use of the library to develop Grid vertical applications in a specific application domain, i.e. the TMA image analysis. Our idea is to develop a system to provide a secure sharing of TMA images, suitable services to search interesting data, computational power to perform the processing and a proper set of functions to actual analyze data.

To achieve these goals, we develop a Grid portal in EGEE, we called it TMAinspect, based on PIMA(GE)^2 Lib software. We used PIMA(GE)^2 Lib to develop a set of functions for useful TMA image analysis, as described in Section 3.5. Starting from these operations, we have to enable their execution in EGEE, furthermore several Grid functionalities have to be developed and integrated in TMAinspect. We use specific services provided by gLite to enable the search of interesting data, and to ensure secure analysis, in particular AMGA and GFAL services. To provide a user friendly framework, we hide all the burdens related with the Grid exploitation and parallel computations through a graphical interface.

The architectural domain is therefore EGEE and gLite, we have implemented a prototype of the portal, it has been tested considering a Grid built with different research institutes in Genoa and Milan. During the remaining of the Section we detail the Grid services developed to overcome the TMA requirements, we describe the general behavior of TMAinspect and we provide an analysis of the performance.

6.6.1 What the Grid can do for TMA technology

The TMA technique has to deal with several difficulties, as storage availability, secure data sharing, metadata handling, and the availability of a tool for image analysis. These topics have been already introduced in Section 3.5, where we provide a solution to overcome the last point, i.e. the development of a suitable and efficient tool to enable TMA analysis. The Grid represents a good opportunity to overcome the other difficulties, however Grid systems remain difficult to exploit for many users, and a simplified use of Grid resources is still a challenge. For this reason we developed a Web interface to hide the complexity of the Grid exploitation. It is aimed to simplify the use of the Grid as well as to enable TMAs handling and analysis.

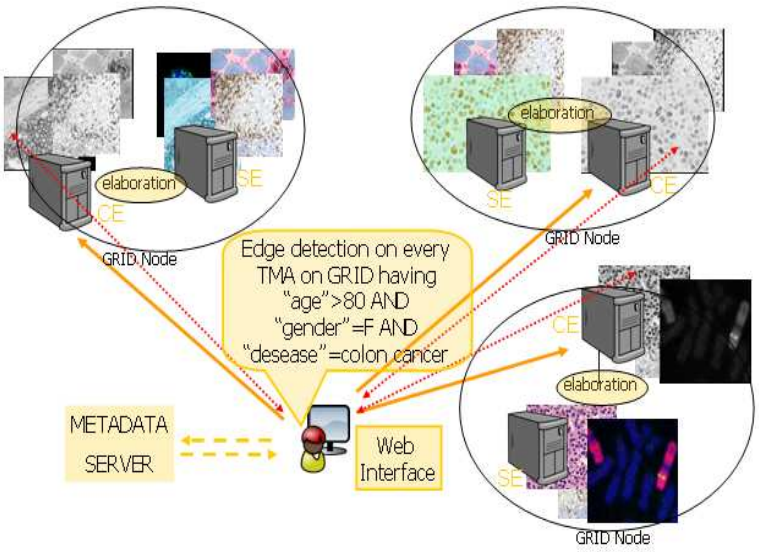


Figure 6.4: *The general scenario we consider during the TMAinspect development.*

Our aim is presented in Figure 6.4, a pathologist has to evaluate some hypotheses regarding the colon cancer in female advanced in years. Thus he/she considers the proper TMA images shared on the grid and requires an edge detection for the images complying with these features. We develop a Grid portal to overcome this situation in a simplified way.

In order to accomplish these tasks, we analyze mainly three aspects: the integration of the specialized set of PIMA(GE)^2 Lib in EGEE, the use of service provided by gLite, the development of a Web interface to enable their use. They represent the main elements of TMAinspect.

The integration of TMA analysis operations in EGEE

As already described in Section 3.5, we develop a set of operations for the analysis of TMA images:

- an edge detection function, extensively used in the analysis of biomedical images;
- a threshold function, which takes an intensity level as input and retrieves a binary image where it is possible to see an area of pixels corresponding to the value;
- a background subtraction function, for removing a constant offset from the signal;
- morphological filters like erosion and dilation kernels, to handle object shapes;
- geometric functions, like rotation, restriction and scale, which allow to better visualize full images or details.

The operations require specific parameters that users have to specify; users can select the operations and provide the parameter through the Web interface of TMAinspect. To execute the jobs gLite provides the JDL, thus for each operation we developed a JDL file containing the parameters to perform the specific function, and the URI of the executable code corresponding to the library function. The JDL file is submitted to the WS GRAM when a user requires the operation, and specifies the parameters. These steps are completely managed by TMAinspect.

The use of services provided by gLite

To enable secure analysis of interesting TMA images we exploit two gLite components: GFAL and AMGA.

GFAL has been considered to ensure a secure access during the data acquisition. Through GFAL, we are able to acquire remote data without an actual transfer on a CE, avoiding multiple copies of sensitive data. This behavior is completely hidden to the users and managed by specific I/O operations, described in Section 6.5 and integrated in TMAinspect.

Considering descriptive information about the data classification, in the TMA context, the term "metadata" refers to four topics: the patient's clinical situation, the sample treatment information, the paraffin block creation and the slide reaction type [139]. A simplified list is given in Table 6.2. As it is possible to see, the metadata does not involve information about patient's identity. We suppose that the institutions owner of biomedical data, already shields such information during the data sharing.

AMGA, ARDA Metadata Grid Application, has been considered to enable metadata management. It provides metadata services for the Grid, i.e. database access services for Grid applications; the aim is to allow user, and user's jobs to discover data describing their

Clinical Data	Sample Data	TMA Block Data	TMA Spot Analysis
Patient Id	Sample Id	Block Id	Spot Id
Birth date	Normal or Pathological desease	Density	xCoordinate
Gender	Extraction type	Creation date	yCoordinate
Ethnicity	Extraction date	Needle diameter	Analisys type
	Site	Image	Label
	Fixation		Searched gene OR protein

Table 6.2: *A simplified list of metadata connected to TMA images that could be considered interesting during the analysis.*

features. AMGA is based on RDBMS, Relational DataBase Management System, and allows the definition of metadata schemes according to user and application needs. Moreover it provides a replication layer which makes databases locally available, and replicate the changes between the different participating databases. AMGA hides database heterogeneity, and refers to the file through the GUID obtained after the insertion into the LFC. AMGA is Grid security compliant since verifies the VO authorization policy during the data access.

Therefore we considered metadata related to TMA images recored in AMGA tables, and we exploit AMGA services to enable the mine of TMA images in order to enable acquisition of such data. Users have to specify data to consider in the analysis, as described in the scenario presented in Figure 6.4, through the Web interface of TMAinspect. The result of the AMGA query is the GUID of the images that correspond to the metadata specified by the users. An example of query to the repository is reported in box code 6.3.

Through the TMAinspect interface, the script reported in box code 6.3 is activated. The script also includes the code to properly visualize the output of the query.

The development of a Web interface

In order to enable in a simple exploitation of Grid resources, we develop a Web interface to accomplish the different steps of the analysis. The Web interface has been adopted for TMAinspect; several screen-shots and the description of their aim are given in the remaining of the Section.

Listing 6.3: *Pseudocode to enable AMGA query.*

```
# parameters of the query: gender, age, gpsequence, disease, ethnicity,
institute. The specif values are stored in variable ``$1'', ...,``$6''

#  possible query #
 query=``selectattr /TMAshare/Analysis:image_id
        ''/TMAshare/PatientInfo:patient_id=/TMAshare/Sample:patient_id
        and /TMAshare/Sample:sample_id=/TMAshare/Analysis:sample_id''

 # ``$1'' = gender
if [ ``$1'' != NULL ]; then
 query1='' and /TMAshare/PatientInfo:gender=\``$1\''''
query=$query$query1
 fi

 # ``$2'' = age
if [ ``$2'' != NULL ]; then
 query2='' and /TMAshare/PatientInfo:age=\``$2\''''
query=$query$query2
 fi

 ...

 #submission of the query and result
echo $query | ./mdclient | egrep '\>␣[A-Za-z0-9]*' > out_query
 echo $query | ./mdclient > out_query
```

6.6.2 TMAinspect

TMAinspect is a user friendly TMA analysis system, it has the purpose to enable the TMA image selection, their parallel processing in EGEE. TMAinspect enables these steps through a Web interface implemented with the Zope Application server framework [141]. Zope is an open source application server for building content management systems, intranets, portals, and custom applications. The system has been developed as a Plone plug in to provide the supported functionalities. Plone is an open source content management system built on top of the Zope application server. Some screenshots of the interface are presented in Figures 6.5, 6.6, and 6.7.

In this Section we provide a description of TMAinspect and its functionalities. We explain the different purpose of the portal considering its Web interface.

The login phase

The creation of a community of certified and authorized people is crucial when dealing with sensible data. This is the reason why a user has to authenticate himself as a member

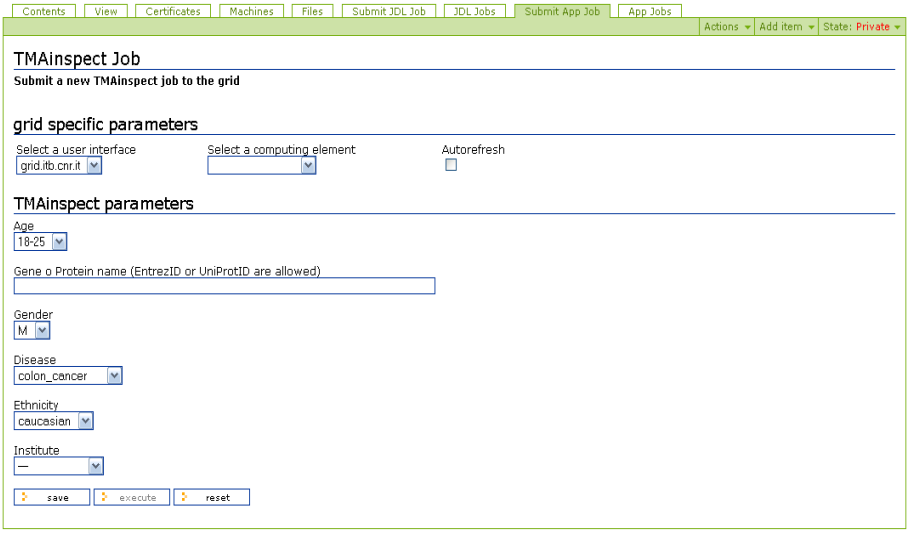


Figure 6.5: *The TMAinspect interface that let the user to specify some demographic a phenotypic parameters.*

of the VO through the TMAinspect interface with user-name and password.

TMA image selection

The next step is the selection of the images of interest through a query on the metadata associated to the images. As already said, in the TMA context, the term "metadata" refers to four topics: the patient's clinical situation, the sample treatment information, the paraffin block creation and the slide reaction.

A query is performed in order to extract the images that satisfy some conditions on metadata, for example the gender and the age of the patients or the disease under study. The interface for the query is presented in Figure 6.5. It corresponds to a query on metadata recorded in AMGA tables. In this way, TMAinspect collects the GUIDs of the images that match the constraints.

The resulting TMA images can be visualized by the user, in order to select the interesting one to process. The framework provides a client-side python-based viewer for remote images, the GUIDs of the images are listed as depicted in Figure 6.6, thus the user to choose the ones to open and process.

Operation selection

Through the TMAinspect interface users have to specify the operations in the sequence of their executions, and their parameters; the TMAinterface to accomplish this task is presented in Figure 6.6.

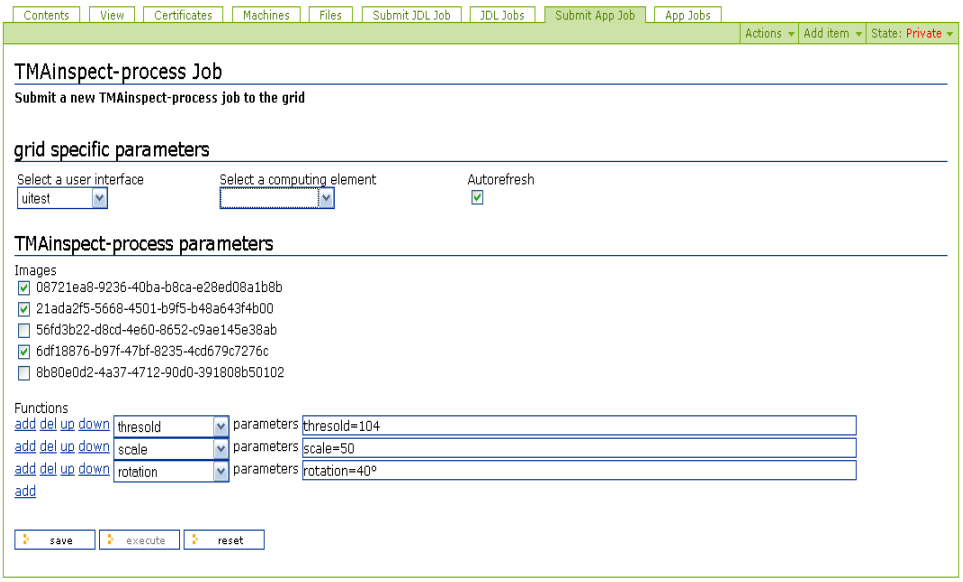


Figure 6.6: *The TMAinspect interface after the query on metadata, the GUIDs of the resulting images are presented. Some of them may be disregarded for further analyses. Through this Web page users have also select the analysis to perform specifying the parameters.*

Thus TMAinspect produces the JDL schema described in the previous Section to enable the execution of the library operations. The JDL contains the parameters to perform the analysis, as well as the URI of the executable codes to run and he GUIDs of the data.

TMA image analysis execution and monitoring

To execute the TMA image analysis, the next step is the selection of the CEs to run the job. The user can express a preference selecting an EGEE resource, in particular the CE. Otherwise the job is submitted to the RB. Each job can be monitored through the TMAinspect interface, as presented in Figure 6.7.

In the job execution, we have two levels of parallelism. The first one is represented by the simultaneous submission of the required operations on all selected images. At this level, the parallelism degree is equal to the number of images to process. The second level of parallelism is given by PIMA(GE)^2 Lib, that enables the efficient execution of parallel computation of each image. In this case, the parallelism degree is determined considering the size of the image to be processed and the complexity of the operations.

TMAinspect produces one JDL file for each image. The only difference among them is the GUID of the image to process; the TMA images are accessed considering the GFAL API, that are automatically employed in TMAinspect when an analysis is required.

Figure 6.7: *TMAinspect interface for job monitoring.*

Output retrieval

The output of the performed through the analysis is automatically retrieved exploiting the traditional EGEE Grid functionality: after the computation the file is stored on a SE and an automatic command line instruction moves it on the UI where the job had been launched. The user can visualize each image by browsing the UI directories and selecting the suitable one containing the output. This image can be downloaded on his local machine without additional limitations because the resulting images are raw data that do not concern sensitive information.

6.6.3 TMAinspect evaluation

We implement a prototype of TMAinspect and we test it considering Computing Elements and Storage Elements at different research institution in Genoa and Milan. The performance of TMAinspect are given by the performance of the single service used in the system, thus the library operations, GFAL and AMGA.

An analysis of the performance of GFAL has been already discussed in the previous Section. The performance of AMGA are based on the effectively of query, and the possibility provided by TMAinspect to elaborate only the most interesting.

Considering the available Computing Elements, TMAinspect can enable a proportional number of TMA image analysis in parallel. The performance of the single analysis have been already described in Section 3.5.

Chapter 7

Conclusions and Future Directions

The objective of this Thesis was the design of high performance tools for image processing; these tools exploit parallel and distributed technologies and represent an attractive solution to enable image processing computing intensive applications.

We presented a parallel library, PIMA(GE)^2 Lib, that provides the commonly used image processing operations. It enables the processing of 2D and 3D data sets, and shields the users from the burdens of parallel computations through a sequential interface. A great attention was paid to the exploitation of advanced parallel tools for data acquisition and to the load balancing of the executions on heterogeneous resources.

Moving towards distributed and Grid environments, we enabled the use of the parallel library as the building blocks to develop complex applications. Therefore we evolved the library into new parallel distributed entities allowing the suitable exploitation of available resources. In the implementation of such entities, we considered as key points: the library code reuse without a re-implementation or modification to it, the definition of user-friendly tools that still provide the library operations hiding their parallelism, the minimization of the overhead related with the exploitation of distributed environments, the preservation of the efficient parallelism of PIMA(GE)^2 Lib, and when it was possible, the exploitation of the scheduling policy applied in PIMA(GE)^2 Lib.

To develop these tools we presented two approaches, the first addresses a client-server interaction in a general distributed environment, while the second studies the use of a parallel image processing library in a Grid environment. In both approaches, we simplified the exploitation of distributed and Grid environments since we provided services that hide the complexity of a direct use of the resources.

In each case, we supposed that the images to process could be stored on a remote host as well as the produced results, that could be loaded on a different remote destination. Hence

remote data transfer were considered exploiting different protocols. The experimentations made with this aim converged in a module for remote I/O, that enriches the efficient I/O operations of PIMA(GE)^2 Lib.

We presented an example of real application of the work to the field of Bioinformatics considering the analysis of TMA images; in this context we performed several steps. The first step was the development of a set of operations through PIMA(GE)^2 Lib to get useful information from the TMA images. The second was the exploitation of the Grid to enable data acquisition complying with a possible restricted access policy defined by the institutions that own TMA data. Other efforts were spent to enable queries on descriptive metadata of the TMA images, and thus to search interesting data shared on the Grid. Last step was the execution of the TMA image analysis on the Grid, preserving the parallelism of the operations and their sequential interface. These works constitute TMAinspect, a Grid framework aimed to the TMA images handling through a Web graphical interface. TMAinspect allows the selection of TMA images through a query on metadata, the selection of the analysis to perform, and the monitoring of the execution on Grid resources.

The works presented in the Thesis can be extended along several directions.

A great interest is in the integration of PIMA(GE)^2 Lib in a Grid virtual laboratory. In fact Grid architectures, considered as the integration of distributed resources, enable the sharing of measurement instrumentation and pervasive large-scale data acquisition devices; this scenario is sometimes called virtual laboratory. Thus a key issue is to provide tools to extract knowledge from the data produced in a virtual laboratory in order to effectively exploit heterogeneous instrumentations and acquisition devices. Considering imaging instruments, PIMA(GE)^2 Lib could provide an effective tool in such scenario, since the different experimentations about the use of the library on the Grid provides encouraging results.

Another research direction is represented by the development of algorithms to analyze TMA images and their reactions with a specific biomarker. Biomarkers are molecules that allow the detection and isolation of a particular cell type, it means that a specific biomarker produces a reaction on particular cells thus it outlines their presence and position in a tissue. For this reason, biomarkers are particularly studied in the cancer research field to detect molecules that may represent a pointer to the presence and the development of a tumor. Thus an interesting application is the identification of the reaction of the tissue with specific biomarkers. We have already started a study about these topics considering colon tissues and a set of five specific biomarkers; the study is made in collaboration with an institution of pathology. The aim is to compare the reaction of a tissue with these set of biomarkers and give back a parameter representing the percentage of hybridization between the targets on the tissue and the probes laying on it. It is not a trivial task, and this research is in a very early stage.

Bibliography

[1] S. Agrawal, J. Dongarra, K. Seymour, and S. Vadhiyar *NetSolve: Past, Present, and Future - a Look at a Grid Enabled Server*, Grid Computing: Making the Global Infrastructure a Reality, Wiley Publisher, 2003.

[2] Altair Web Site, `http://www.pbsgridworks.com/`

[3] S.R. Amendolia, F. Estrella, T. Hauer, D. Manset, R. McClatchey, M. Odeh, T. Reading, D. Rogulin, D. Schottlander, and T. Solomonides, *Grid Databases for Shared Image Analysis in the MammoGrid Project*, In IEEE Proceedings of IDEAS'04, pag 302 - 311, 2004.

[4] K. Amin, G. von Laszewski, M. Hategan, R. Al-Ali, O. Rana, and D. Walker, *An abstraction model for a Grid execution framework*, Journal of Systems Architecture, vol. 52, issue 2, pag. 73-87, 2006.

[5] E. Andersin, Z. Bai, C. Buschof, J. Demmel, J.J. Dongarra, J. DuCroz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen, *LAPACK Users' Guide, 2nd ed.*, SIAM, Philadelphia, PA, 1995.

[6] G. Andronico, V. Ardizzone, R. Barbera, R. Catania, A. Carrieri, A. Falzone, E. Giorgio, G. La Rocca, S. Monforte, M. Pappalardo, G. Passaro, and G. Platania, *GILDA: The Grid INFN Virtual Laboratory for Dissemination Activities*. Procs of the First International Conference on Testbeds and Research Infrastructures for the DEvelopment of NeTworks and COMmunities (TRIDENTCOM'05), pag. 304-305, 2005

[7] D. Arnold, S. Agrawal, S. Blackford, J.J. Dongarra, M. Miller, K. Seymour, K. Sagi, Z. Shi, and S.Vadhiyar. *Users' Guide to NetSolve V1.4.1*, Innovative Computing Dept. Technical Report, University of Tennessee, ICL-UT-02-05, 2002.

[8] D. Arnold, and J. Dongarra, *Developing an Architecture to Support the Implementation and Development of Scientific Computing Applications*, FIP Conference Proceedings, Vol. 188, pag. 39-56, 2000.

[9] The AstroGrid homepage, http://www2.astrogrid.org/

[10] H. Attiya, J. Welch. *Distributed Computing: Fundamentals, Simulations*, and Advanced Topics. Wiley-IEEE, 2004.

[11] B. Bacci, M. Danelutto, S. Pelagatti, and M. Vanneschi, *SkIE: A heterogeneous environment for HPC applications*, Parallel Computing, vol 25, issue 13-14, pag. 1827-1852, 1999.

[12] M. Baker. *Cluster Computing White Paper*, 2000, available at http://arxiv.org/abs/cs/0004014

[13] S. Balay, W. Gropp, L. Curfman McInnes, and Barry Smit, *PETSc 2.0 Users' Manual*, Technical Report Argonne National Laboratory-95/11 revision 2.0.17, 1996.

[14] R.Barbera, *The GENIUS Grid Portal*, Computing inHigh Energy and Nuclear Physic, 28-28, 2003.

[15] J. Bilmes, K. Asanovic, C.-W. Chin, and J. Demmel, *Optimizing Matrix Multiply Using PHiPAC: A Portable, High-Performance, ANSI C Coding Methodology*, Proceedings of the International Conference on Supercomputing, pp. 340-347, 1997.

[16] L. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J.J. Dongarra, S. Hammarling, G. Henry, A. Petit, K. Stanley, D. Walker, and R. Whaley. *ScaLAPACK Users' Guide*, SIAM, Philadelphia, PA, 1997.

[17] C. Boeres, and V.E.F. Rebello, *EasyGrid: towards a framework for the automatic Grid enabling of legacy MPI applications*, Concurrency and Computation: Practice & Experience, Volume 16, Issue 5, pag. 425-432, 2004.

[18] P. Boulet, J. Dongarra, Yves Robert, and F. Vivien. *Static tiling for heterogeneous computing platforms*. Parallel Computing, Vol. 25, Issue 5, pag. 547-568, 1999.

[19] R. Buyya, D. Abramson, and J. Giddy, *Nimrod/G: An Architecture of a Resource Management and Scheduling System in a Global Computational Grid*, IEEE Procedeeng of HPC Asia 2000, pag. 283-289, 2000.

[20] caBIG Community Website, https://cabig.nci.nih.gov/

[21] caGrid Home Page, https://cabig.nci.nih.gov/workspaces/Architecture/caGrid

[22] A. Ching, K. Coloma, J. Li, and A. Choudhary, *High-performance techniques for parallel I/O*. In Handbook of Parallel Computing: Models, Algorithms, and Applications, CRC Press, (2007).

[23] J. Choi, J.J. Dongarra, S. Ostrouchov, A. Petit, D. Walker, and R. Whaley. *A Proposal for a Set of Parallel Basic Linear Algebra Subprograms*, Technical Report UT CS-95-292, LAPACK Working Note#100, University of Tennessee, 1995.

[24] J. Chung, and J. Son, *A CORBA-Based Telemedicine System for Medical Image Analysis and Modeling*, IEEE Proceedings of the 14th Symposium on Computer-Based Medical Systems, 2001.

[25] A. Clematis, and A. Corana, *Modeling performance of heterogeneous parallel computing systems*, Parallel Computing, vol. 25, pag. 1131-1145, 1999.

[26] A. Clematis, A. Corana, D. D'Agostino, V. Gianuzzi, and A. Merlo, *Resource Selection and Application Execution in a Grid: A Migration Experience from GT2 to GT4*. Procs. of GADA 2006, LNCS 4276, pag. 1132-1142, Springer, 2006.

[27] A. Clematis, D. D'Agostino, and A. Galizia, *An Object Interface for Interoperability of Image Processing Parallel Library in a Distributed Environment*. In Proceedings of ICIAP 2005, LNCS 3617, pp. 584-589, 2005.

[28] A. Clematis, D. D'Agostino, and A. Galizia, *A Parallel IMAGE processing Server for Distributed Applications*. Parallel Computing: Concurrent & Future Issues of High-End Computing, John von Neumann Institute for Computing (NIC) series, vol. 33, pag. 607-614, 2005.

[29] A. Clematis, D. D'Agostino, and A. Galizia, *Parallel I/O Aspects in* PIMA(GE)^2 *Lib*. In: Parallel Computing: Architectures, Algorithms and Applications, Proceedings of the ParCo 2007, Advances in Parallel Computing, Volume 15, pag. 441-448, IOS Press, 2007.

[30] A. Clematis, D. D'Agostino, and A. Galizia, *The use of* PIMA(GE)^2 *Lib for efficient image processing in a Grid environment*, accepted for the publication in the Proceeding of INGRID 2007.

[31] A. Clematis, G. Dodero, and V. Gianuzzi, *Efficient use of parallel libraries on heterogeneous networks of workstations*, Journal of Systems Architecture, vol. 46, pp. 641-653, 2000.

[32] Condor Project Homepage, `http://www.cs.wisc.edu/condor/`

[33] The CORBA Homepage, `http://www.corba.org/`

[34] E. de Moraes Barros Jr., M. Sehn, A. von Wangenheim, and D. Krechel, *A model for distributed medical image processing using CORBA*, IEEE Proceedings of the 14th Symposium on Computer-Based Medical Systems, 2001.

[35] D. P. del Rey, J. Crespo, A. Anguita, J.L.P. Ordonez, J. Dorado, G. Bueno, V. Feliu, A. Estruch, and J.A. Heredia, *Biomedical Image Processing Integration Through INBIOMED: A Web Services-Based Platform*, LNCS 3745, pag. 34-43, 2005.

[36] T.A. DeFanti, I. Foster, M.E. Papka, and R. Stevens. *Overview of the I-WAY: Wide Area Visual Supercomputing*, the International Journal of Supercomputer Applications and High Performance Computing, Vol. 10, Issue 2-3, pag. 123-131, 1996.

[37] T. Delaitre, T. Kiss, A. Goyeneche, G. Terstyanszky, S. Winter, and P. Kacsuk, *GEMLCA: Running Legacy Code Applications as Grid Services*, Journal of Grid Computing, Vol. 3, Issue 1-2, Springer, pag. 75-90, 2005.

[38] A. Dennis, C. Peréz and T. Priol, *PadicoTM: An open integration framework for communication middleware and runtimes*, IEEE Intl. Symposium on Cluster Computing and Grid, 2002.

[39] The distributed.net project, `http://www.distributed.net/`

[40] J.J. Dongarra, G. Bosilca, Z. Chen, V. Eijkhout, G. Fagg, E. Fuentes, J. Langou, P. Luszczek, J. Pjesivac-Grbovic, K. Seymour, H. You, and S. Vadhiyar, *Self-adapting numerical software (SANS) effort*, IBM Journal of Research and Development, Vol. 50, Issue 2/3, pag. 223-238, 2006.

[41] J.J. Dongarra, J. Bunch, C. Moler, and G. Stewart, *LINPACK Users' Guide*, SIAM, Philadelphia, PA, 1979.

[42] J.J. Dongarra, J. DuCroz, I. Duff, and S. Hammarling, *A Set of Level 3 Basic Linear Algebra Subprograms*, ACM Trans. Math. Softw., 16:1-17, 1990.

[43] J.J. Dongarra, J. DuCroz, S. Hammarling, R. Hanson. *An Extended set of Fortran Basic Linear Algebra Subprograms*, ACM Trans. Math. Softw., 14:1-17, 1988.

[44] J.J. Dongarra, I. Duff, D.C. Sorensen, and H.A. van der Vorst, *Numerical Linear Algebra on High-Performance Computers*, SIAM, 1998.

[45] J.J. Dongarra, I. Foster, G. Fox, K. Kennedy, W. Gropp, L. Torczon, and A. White, *The Sourcebook of parallel computing*, Morgan Kaufmann, 2002.

[46] J.J. Dongarra, and R. Whaley, *A Users' Guide to the BLACS v1.0*, Technical Report UT CS-95-281, LAPACK Working Note#94, University of Tennessee, 1995.

[47] E. Dolan, R. Fourer, J. J. More, and Munson Munson, *The NEOS server for optimization: Version 4 and beyond*, Technical Report ANL/MCS-P947-0202, Mathematics and Computer Science Division, Argonne National Laboratory, 2002.

[48] EGEE Home Page, `http://www.eu-egee.org/`

[49] D.H.J. Epema, M. Livny, R v.Dantzig, X. Evers, and J. Pruyne, *A Worldwide Flock of Condors: Load Sharing among Workstation Clusters*, Journal on Future Generation of Computer Systems, Vol. 12, pag. 53-65, 1996.

[50] I. Forster. *The Nexus Approach to Integrating Multithreading and Communication*, Journal of Parallel and Distributed Computing, Vol. 37, Issue 1, pag. 70-82, 1996.

[51] I. Foster and C. Kesselman, *Globus: A metacomputing infrastructure toolkit*, Intl J. Supercomputer Applications, Vol. 11, Issue 2, pag. 115-128, 1997.

[52] I. Foster and C. Kesselman, *The grid: blueprint for a new computing infrastructure*, 2nd Edition, Morgan Kaufmann Publishers Inc., 2004.

[53] I. Foster, C. Kesselman and S. Tuecke, *The anatomy of the Grid: enabling scalable Virtual Organizations*, International Journal High-Performance Computing Applicationd, Vol. 15, Issue 3, 2001.

[54] I. Foster, C. Kesselman, J. Nick and S. Tuecke, *The Physiology of the Grid: an open Grid services architecture for distributed systems integration*, Open grid service infracstrutture WG, Global Grid Forum, June, 2002.

[55] M. Frigo, S. G. Johnson. *FFTW: An Adaptive Software Architecture for the FFT*, Proceedings of the IEEE International Conference on Acoustics, Speech, and Signal Processing, pag. 1381-1384, 1998.

[56] A. Galizia, F. Viti, D. D'Agostino, I. Merelli, A. Clematis, and L. Milanesi, *Experimenting Grid Protocols to Improve Privacy Preservation in Efficient Distributed Image Processing*, In: Parallel Computing: Architectures, Algorithms and Applications, Proceedings of the ParCo 2007, Advances in Parallel Computing, Vol. 15, pp. 139-146, IOS Press, 2007.

[57] A. Galizia, F. Viti, A. Orro, D. D'Agostino, I. Merelli, A. Clematis, and L. Milanesi, *Enabling parallel TMA image analysis in a Grid environment*, accepted for the publication in the Proceeding of 3PGIC 2008.

[58] A. Galizia, F. Viti, A. Orro, D. D'Agostino, I. Merelli, A. Clematis, and L. Milanesi, *TMAinspect, an EGEE Framework for Tissue MicroArray image handling*, accepted for the publication in the Proceeding of CCGRID 2008.

[59] C. Germain, V. Breton, P. Clarysse, Y. Gaudeau, T. Glatard, E. Jeannot, Y Legré, C. Loomis, I. Magnin, J. Montagnat, J.M. Moureaux, A. Osorio, X. Pennec, and R. Texier, *Grid-enabling medical image analysis*, Journal of Clinical Monitoring and Computing, Vol. 19, Num. 4-5, pag. 339-349, 2005.

[60] Ligthweight Middleware for Grid Computing, http://glite.web.cern.ch/glite/

[61] The Globus Alliance, http://www.globus.org/

[62] The Globus Toolkit Homepage, http://www.globus.org/toolkit/

[63] T. Goodale, G. Allen, G. Lanfermann, J. Masso, T. Radke, E. Seidel, and J. Shalf. *The cactus framework and toolkit: Design and applications*, In Proceeding of VECPAR 2002, LNCS 2565, pag. 15-36, Springer, 2003.

[64] The Great Internet Mersenne Prime Search, http://mersenne.org/

[65] The Grid Application Development Software Project (GrADS), http://hipersoft.rice.edu/grads/index.htm

[66] GridLab, The GridSphere Portal, http://www.gridsphere.org

[67] A. Grimshaw, *Metasystems: An Approach Combining Parallel Processing and Heterogeneous Distributed Computing Systems*, Journal of Parallel and Distributed Computing, Vol. 21, Issue 3, pag. 257-270, 1994.

[68] A.S. Grimshaw, and W.A. Wulf, *Legion Flexible Support for Wide-Area Computing*, HPDC, 89-99, 1996.

[69] Y. Huang, I. Taylor, D.W. Walker, and R. Davies, *Wrapping Legacy Codes for Grid-Based Applications*, IEEE Proceeding of IPDPS'03, p.139.2, 2003.

[70] International Science Grid This Week, http://www.isgtw.org/

[71] A.A. Khokhar, V.K. Prasanna, M.E. Shaaban, and C.L. Wang, *Heterogeneous computing: challenges and opportunities*, Computer, vol. 26, no. 6, pp. 18-27, IEEE Computer Society, 1993.

[72] C.L. Lawson, R.J. Hanson, D. Kincaid, and F. T. Krogh, *Basic Linear Algebra Subprograms for FORTRAN usage*, ACM Trans. Math. Soft., Vol 5, pag. 308-323, 1979.

[73] J. Lebak, J. Kepner, H. Hoffmann, and E. Rudtledge, *Parallel VSIPL++: an open standard library for high-performance parallel signal processing*. IEEE Proceedings, vol. 93-2. February 2005

[74] M. Li, O.F. Rana, and D.W. Walker, *Wrapping MPI-based legacy codes as Java/CORBA components*, Future Generation Computer Systems, Vol. 18, Num. 2, pag. 213-223, 2001.

[75] M. Li, D.W. Walker, O.F. Rana, and Y. Huang, *Migrating legacy codes to distributed computing enviroments:a CORBA approach*, Information and Software Technology Vol. 46, Issue 7, Pag. 457-464, 2004.

[76] K. Liakos, A. Burger, and R. Baldock, *Distributed Processing of Large BioMedical 3D Images*, LNCS 3402, pages 142-155, 2005.

[77] J. Mellor-Crummey, P. Beckman, K. Cooper, J. Dongarra, W. Gropp, E. Lusk, B. Miller, and K. Yelick, *Creating Software Tools and Libraries for Leadership Computing*, CTWatch Quarterly, Volume 3, Number 4, 2007.

[78] J. Montagnat, V. Breton, and I.E. Magnin, *Using Grid Technologies to Face Medical Image Analysis Challenges*, In Biogrid'03, IEEE Proceedings of CCGrid03, 2003.

[79] J. Montagnat, A. Frohner, D. Jouvenot, C. Pera, P. Kunszt, B. Koblitz, N. Santos, C. Loomis, R. Texier, D. Lingrand, P. Guio, R.B. Da Rocha, A.S. de Almeida, Z. Farkas, *A Secure Grid Medical Data Manager Interfaced to the gLite Middleware Journal Journal of Grid Computing*, to appear in Journal of Grid Computing, Springer, 2007.

[80] M. S. Müller, M. Hess, and E. Gabriel, *Grid enabled MPI solutions for Clusters*. Procs of the 3rd IEEE International Symposium on Cluster Computing and the Grid (CCGrid 2003), pag. 18-25, 2003

[81] MyGrid Home Page, `http://www.mygrid.org.uk/`

[82] H. Nakada, S. Matsuoka, K. Seymour, J. Dongarra, C. Lee, and H. Casanova, *Overview of GridRPC: A Remote Procedure Call API for Grid Computing*, LNCS 2536, pag 274-278, 2002.

[83] K.Keathey and D. Gannon, *PARDIS: a parallel approach to CORBA*, Supercomputing'97, ACM/IEEE, November 1997.

[84] A.A. Khokhar, V.K. Prasanna, M.E. Shaaban, and C.L. Wang. *Heterogeneous computing: challenges and opportunities*, Computer, Vol. 26, Issue 6, pag. 18-27, 1993.

[85] D. Koelma, E. Poll, F. Seinstra. *Horus (Relise 0.9.2)*, Technical Report, Intelligent Sensory Information System, Faculty of Science, University of Amsterdam, 2000.

[86] J. Kononen, L. Bubendorf, A. Kallioniemi, M. Barlund, P. Schraml, S. Leighton, J. Torhorst, M.J. Mihatsch, G. Sauter, and O.P Kallioniemi, *Tissue microarrays for high-throughput molecular profiling of tumor Specimens*, Nature Medicine, Vol. 4, pag. 844-847, 1998.

[87] Mercury Computer System, Inc. and Objective Interface System, Inc. and MPI Software Technology, Inc. and Los Alamos National Laboratory. *Data Parallel CORBA*, Initial Submission August 2000.

[88] S. Mousses, L. Bubendorf, U. Wagner, G. Hostetter, J. Kononen, R. Cornelison, N. Goldberger, A.G. Elkahloun, N. Willi, P. Koivisto, W. Ferhle, M. Raffeld, G. Sauter, and O.P. Kallioniemi, *Clinical validation of candidate genes associated with prostate cancer progression in the CWR22 model system using tissue microarrays*, Cancer Research, Vol. 62, pag. 1256-1260, 2002.

[89] MPICH2 Home Page, `http://www-unix.mcs.anl.gov/mpi/mpich2/`

[90] MPI Forum Home Page, `http://www.mpi-forum.org/`

[91] NCSA Alliance Scientif Portal Project, `http://www.exetreme.indiana.edu/alliance`

[92] Network for Earthquake Engineering Simulation Grid (NEESgrid), `http://www.neesgrid.org`

[93] C. Nicolescu, P. Jonker, *EASY-PIPE an easy to use parallel image processing environment based on algorithmic skeletons*, In IEEE Proceedings of 15th IPDPS, (2001).

[94] Object Management Group, *Request For Proposal: Data Parallel Application Support for CORBA*, March 2000.

[95] Open Grid Forum, `http://www.ogf.org/`

[96] P. Oliveira, and H. du Buf, *SPMD Image Processing on Beowulf Clusters: Directives and Libraries*, In IEEE Proceedings of 7th IPDPS, 2003.

[97] The Open Software Foundation, `http://www.opengroup.org/dce/`

[98] OMG Official Website, `http://www.omg.org`

[99] Parallel Virtual File System Version 2, `http://www.pvfs.org`

[100] A. Pepitet, S. Blackford, J. Dongarra, B. Ellis, G. Fagg, K. Roche and S. Vadhiyar, *Numerical libraries and the Grid*, International Journal of High Performance Applications and Supercomputing, 15-4, 2001.

[101] C. Peréz, T. Priol, and A. Ribes, *PACO++: A Parallel Object Model for High Performance Distributed Systems*, IEEE Proceedings of HICSS'04, Tack 9, p. 90274, 2004.

[102] D. Petcu, and V. Iordan, *Grid Service Based on GIMP for Processing Remote Sensing Images*, In IEEE Proceeding of SYNASC'06, pag. 251-258, 2006.

[103] PETSc Web page, `http://www.mcs.anl.gov/petsc`

[104] Platform Computiong Web Site, `http://www.platform.com`

[105] T. Priol, and C. Ren, *Cobra: a CORBA-compliant programming environment for High-Performance computing*, Euro-Par'98, September 1998.

[106] G. Ritter, and J. Wilson *Handbook of Computer Vision Algorithms in Image Algebra*, 2nd Edition, CRC Press Inc., 2001.

[107] C. René, and T. Priol, *MPI code encapsulating using parallel CORBA object*. Cluster Computing, Vol. 3, Issue 4, pag. 255-263, 2000.

[108] K. J. Roche, and J. J. Dongarra, *Deploying Parallel Numerical Library Routines to Cluster Computing in a Self- Adapting Fashion*, Parallel Computing: Advances and Current Issues, Imperial College Press, London, 2002.

[109] ROMIO home page, `http://www.mcs.anl.gov/romio`

[110] R. Ross, R. Thakur, A. Chouldhary.*Achievements and challenges for I/O in computational science*, Journal of Physics: Conference Series 16, pag. 501-509, SciDAC 2005.

[111] M. Sawinska, D. Kurzyniec, J. Sawinski and V. Sunderam, *Automated De- ployment Support for Parallel Distributed Computing*. In the Proceedings of PDP 2007, pag. 139-146, IEEE Computer Society, 2007.

[112] F.J. Seinstra, *User Transparent Parallel Image Processing*, PhD thesis, Intelligent Sensory Information System, Faculty of Science, University of Amsterdam, 2003.

[113] F.J. Seinstra, and D. Koelma, *User Transparency: A Fully Sequential Programming Model for Efficient Data Parallel Image Processing*, Concurrency and Computation: Practice & Experience, Vol 16, Issue 6, pag. 611-644, 2004.

[114] F.J. Seinstra, and D. Koelma, *P-3PC: A Point-to-Point Communication Model for Automatic and Optimal Decomposition of Regular Domain Problems*, IEEE Transactions on Parallel and Distributed Systems,Vol. 13, Issue 7, pag. 758-768, 2002.

[115] F.J. Seinstra, and D. Koelma, *Lazy Parallelization: A Finite State Machine Based Optimization Approach for Data Parallel Image Processing Applications*, IEEE Proceedings of the 17th IPDPS, 2003.

[116] F. Seinstra, D. Koelma, and J.M. Geusebroek, *A Software Architecture for User Transparent Parallel Image Processing*, Parallel Computing, Vol. 28, Issue 7-8, 2002.

[117] SETIhome project, `http://setiathome.ssl.berkeley.edu/`

[118] Z. Shen, J. Luo, G. Huang, D. Ming, W. Ma, and H. Sheng, *Distributed computing model for processing remotely sensed images based on grid computing*, Information Sciences, Vol. 177, Issue 2, pag. 504-518, 2007.

[119] R. Sinnott, M. Bayer, D. Houghton, D. Berry, M. Ferrier, *Bridges: Security Focused Integration of Distributed Biomedical Data*, Proceedings of UK e-Science All Hands Meeting 2003.

[120] R. Sinnott, C. Bayliss, J. Jiang, *Security-oriented Data Grids for Microarray Expression Profiles*, Proceedings of HealthGrid2007, 2007.

[121] G. Sipos, and P. Kacsuk, *Multi-Grid, Multi-User Workflows in the P-GRADE Portal*. Journal of Grid Computing, vol. 3, no. 3-4, Kluwer Academic Publishers, pag. 221-238, 2006.

[122] D. Skillicorn, D. Talia. *Models and Languages for Parallel Computation*, ACM Computing Survey, Vol. 30, Issue 2, pag. 123-169, 1998.

[123] H.M. Sneed, *Encapsulation of Legacy Software: A technique for reuse software components*. Annals of Software Engineering, Vol 9, pag. 293-313, 2000.

[124] H.M. Sneed, *Integrating legacy software into a service architecture*, IEEE Proceedings of CSMR'06, pag. 3-14, 2006.

[125] L. Smarr,and C.E.Carlett, *Metacomputing*, Communications of the ACM, Vol. 35, Issue 6, pag. 44-52. 1992.

[126] B.T Smith, J.M. Boyle, J.J. Dongarra, B.S. Garbow, Y. Ikele, V. Klema, and C. Moler, *Matrix Eigensystem Routines - EISPACK Guide, 2nd ed.*, vol. 6, Springer-Verlag, Yew York, 1976.

[127] D.C. Schimdt, D.L. Levine and S. Mungee, *The design of the TAO Real-Time Object Request Broker*, Computer Communications, Elsivier Science, Vol. 21 n. 4, April 1998.

[128] R. Stevens, P.Woodward, T. DeFanti, and C. Catlett. *Fomr I-WAY to the National Technology Gird*, Communication of the ACM, Vol. 40, Issue 11, pg. 51-60, 1997.

[129] J.M. Squyres, A. Lumsdaine, and R.L. Stevenson, *A Toolkit for Parallel Image Processing*, in Parallel and Distributed Methods for Image Processing II, Proc. SPIE, Vol 3452, 1998.

[130] Sun Microsystems Inc., *NFS: Network File System Version 3 Protocol Specification*, Sun Microsystems Inc., Mountain View,CA, (1993).

[131] C. Szyperski, *Component Software: Beyond Object-Oriented Programming*, ACM Press, New York, 1999.

[132] Y. Tanaka, H. Nakada, S. Sekiguchi, Suzumura Suzumura, and S. Matsuoka. *Ninf-G: A reference implementation of RPC-based programming middleware for Grid computing*. Journal of Grid Computing, Vol. 1, Issue 1, pag. 4151, 2003.

[133] A. Tanenbaum and M. Van Steen, *Distributed Systems: Principles and Paradigms*, Prentice Hall, Pearson Education, USA, 2002.

[134] I. Taylor, I. Wang, M. Shields, and S. Majithia,*Distributed computing with Triana on the Grid*, Concurrency and Computation:Practice and Experience, Vol 17, Issue 9, pag. 1197 - 1214, 2005.

[135] R. Thakur, W. Gropp, and E. Lusk, *Optimizing Noncontiguous Accesses in MPI-IO*, Parallel Computing, Vol 28, Issue 1, pag. 83-105, 2002.

[136] VGrADS Web page, http://vgrads.rice.edu/

[137] S. Vinoski, *CORBA: Integrating Diverse Applications Within Distributed Heterogeneous Environments*, IEEE Communications Magazine, 14-2-1997.

[138] The Visible Human Project, `http://www.nlm.nih.gov/research/visible/visible\_human.html`

[139] F. Viti, I. Merelli, A. Caprera, B. Lazzari, and L. Milanesi, *Ontology-based image description oriented to Tissue MicroArray experiments design*, accepted for the publicaton on BMC Bioinformatics, 2008.

[140] N. Wang, D.C. Schmitdt and D. Levine, *An overview of the CORBA component model*, Addison-Wesley, 2000.

[141] The Web Site for the Zope Community, `http://www.zope.org`

[142] Wikipedia, the free encyclopedia that anyone can edit, `http://en.wikipedia.org/wiki/Main\_Page`

[143] R. Whaley, A. Petit, and J.J. Dongarra, *Automated Empirical Optimization of Software and the ATLAS Project*, Parallel Computing, Vol 27, n 1-2, pp 3-35, 2001.

[144] R. Williams, *Grid Computing: Making the Global Infrastructure a Reality*, In Communications Networking & Distributed Systems, Wiley Series, 2003.