John von Neumann Institute for Computing

A Parallel IMAGE Processing Server for Distributed Applications

A. Clematis, D. D'Agostino, A. Galizia

published in

Parallel Computing:
Current & Future Issues of High-End Computing,
Proceedings of the International Conference ParCo 2005,
G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, E. Zapata (Editors),
John von Neumann Institute for Computing, Jülich,
NIC Series, Vol. **33**, ISBN 3-00-017352-8, pp. 607-614, 2006.

http://www.fz-juelich.de/nic-series/volume33

A Parallel IMAGE Processing Server for Distributed Applications

A. Clematis[a], D. D'Agostino[a], A. Galizia[a]

[a]IMATI-CNR, Genova, Italy

The use of parallel libraries for image processing is a common practice in the implementation of monolithic applications. Nowadays there is an increasing interest in moving towards distributed and heterogeneous applications also in the image processing community. The computational Grid may represent an adequate middleware support and infrastructure at this purpose. A problem of interest in this context is the interoperability and reuse of available parallel libraries, possibly through a dynamic interaction model that permits to accommodate request arriving from heterogeneous clients in a distributed environment. In this paper we present $\mathrm{PIMA(GE)}^2$, a Parallel IMAGE processing GEnova server obtained wrapping up a legacy code parallel library. The parallel server is implemented using CORBA and may be integrated in Grid architecture. The $\mathrm{PIMA(GE)}^2$ server organization and its interaction with user requests are discussed. Early experimental results are presented.

1. Introduction

In the era of distributed and Grid computing [6], the concept of library is evolving from a static software entity to a more dynamic one. A library is often considered as a static tool, mainly designed to be executed on an homogeneous architecture in order to develop monolithic applications. Moving towards distributed, dynamic and heterogeneous environments, it is transformed in a component-based bundle that can be accessed using a server based interaction. This new entity can be used to develop adaptive and distributed applications that are designed using a component based approach and executed in a changing and heterogeneous environment. It means that the functions performed by the library are evolved into components, and an application becomes a dynamic components concatenation, executed on an heterogeneous and distributed infrastructures. The server should be able to manage multiple requests performed by different distributed clients.
The integration of parallel libraries in heterogeneous and Grid-based environment is a topic of interest in the scientific community. The use of sequential and parallel libraries is well assessed for numerical applications [1], but also in other fields like image processing it is now possible to find efficient and effective libraries. These libraries permit to speed-up the development process and often provide optimized performances on a wide range of target architectures. Their quality can be assessed using a well defined set of requirements that permits to verify the library effectiveness and efficiency. These requirements include:

- easy of use, the most compelling need for the users is a simple and natural tool to exploit the operations allowed, therefore a good point is represented by a well designed interface;
- transparency to the parallelism and the optimization policy, it is essential to provide tools that shield their users from the intrinsic complexities of parallel computations;
- efficiency, the users should be capable of obtaining significant performance gains in the most common image processing operations;
- portability, it is essential to ensure executions on different target machines, especially with the new emerging technologies and architecture;

- robustness, it is essential to provide computations that are insensitive to the variations of the data and ensure correct results;
- scalability, a well-designed software architecture has to increase its performance under the different used technologies, especially when resources are added;
- completeness, the users have not the necessity of using other packages related with the environment or the parallelism.

Considering the level of quality and the development cost of image processing parallel libraries it is a great interest their interoperability and reuse in distributed and heterogeneous environments. At this purpose in the recent past different technologies and methodologies have been developed in order to obtain interoperability and reuse of legacy code, one of the best suited methodology is code encapsulation [16], while a suitable technology is represented by the CORBA framework [18].
Our goal is to permit interoperability and reuse of a parallel image processing library in an heterogeneous environment obtaining at the same time a high level of flexibility and a dynamic behaviour permitting to develop Client Server image processing applications that exploit performance figures provided by an optimized parallel library and efficiently utilize the underlying distributed infrastructure.
To achieve this goal we have designed and implemented $\mathrm{PIMA(GE)^2}$, a Parallel IMAGE Processing GEnova Server [2], obtained by wrapping an image processing legacy code using CORBA [3]. The computational kernel is provided by a C++/MPI-2 parallel library, exploiting the algorithmic patterns that characterize image processing operations. This library is considered, for the purpose of the definition of the server as a legacy code. In order to permit code encapsulation an adequate object hierarchy is adopted without any actual change to the legacy code; its aim is to drive the definition of an adequate CORBA $\mathrm{PIMA(GE)^2}$ Application Programming Interface (API). One the most important problem in the design of the internal $\mathrm{PIMA(GE)^2}$ structure is the embedding of C++/MPI-2 computations within a CORBA object with a limited overhead. The focus of the paper is on this aspect and the adopted design solutions and early experimental results are presented. It is important to notice that our solution is based on standard CORBA implementation, according to the Object Management Group, (OMG) [11], specification.
Different related works addresses similar goals. In the field of image processing an increasing attention is paid to parallel processing. In fact we find different and actual examples of parallel library, for example VSIPL++ [9], ParHorus [15], PIPT [12]; they provide object-oriented image processing code, and ensure high performance executions. They are compliant with the requirements defined in the list presented above in this Section. The use of CORBA in order to wrap parallel applications has been considered in different works [4], [13], [5]; it is obtained through the definition of CORBA compliant parallel objects and environments. Finally the problem of library integration in distributed and Grid based architectures has been considered by different projects like Netsolve [10] and GrADS [8] .
In this paper we shortly outline the library interface and then we provide a more detailed description of the $\mathrm{PIMA(GE)^2}$ internal organization and behaviour.

2. Parallel image processing library structure

The main goal of the $\mathrm{PIMA(GE)^2}$ server is the encapsulation of C++/MPI-2 parallel library in order to use it in a distributed environment. The library implements a large set of the most common image processing operations, according to the classification and the organization provided in Image

Algebra, [14]. The computation is data parallel, with a coordinator process during the initialization and the I/O phases; the parallelism is transparent to the user and totally managed by the library. Also the optimization policy applied in the library is transparent; they are both hidden through a sequential API. The optimization policy is oriented to different levels: to perform an opportune management of communication and memory operations; data distribution is aimed to obtain load balancing among the MPI processes. The library itself is an ongoing work, in fact other efforts are spent to obtain an adaptive code [7]; we are interested in designing code that operate efficiently on different target architectures.
Focusing on the provided operations of the library, we can group them in conceptual objects, that are not intended to prescribe how an operation is performed but to underline the operation similarity and to help in the definition of an effective and efficient interface. The conceptual objects allow an easier management of the library operations, since the user is no longer involved with a large number of functions, but he has to consider and handle a small set of clear and well-defined objects. The operations are grouped together according to different rules, such as the nature similarity, or the data structures processed. In this way we outlined in the library code eight objects: I/O Operations, Point Operations, Image Arithmetic, Reduce Operations, Geometric Operations, Neighborhood Operations, Morphology Operations, Differential Operations. We considered a hidden code level, it is not included in the library code classification; the library hidden code is concerned with the management of secondary data structures, with the parallelism and the optimization policy.

3. The library hierarchy and the $PIMA(GE)^2$ IDL interface

To allow a simple and coherent use of the library in a distributed environment, the most relevant element is the definition of effective and flexible interfaces, that permit the development of efficient image processing applications. It can be achieved through a natural evolution from the library legacy code to the $\mathrm{PIMA(GE)}^2$ server; for this reason we exploit the conceptual model introduced in the previous Section, and define a logical hierarchy of image processing operations and objects. Its organization is presented in Figure 1. The aim of the object hierarchy is to drive the definition of an adequate CORBA interface, and its Interface Description Language (IDL) based implementation. It represents the $\mathrm{PIMA(GE)}^2$ Application Programming Interface (API).

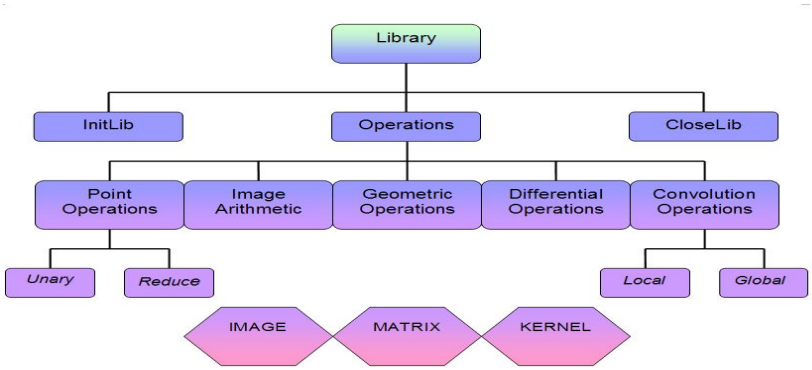


Figure 1. Overview of the $\mathrm{PIMA(GE)}^2$ hierarchy

3.1. The basic elements of the hierarchy

The basic elements of the hierarchy are represented by the main data structures of the library: IMAGE, used to store images; KERNEL, used to store convolution kernels; and MATRIX, used to store geometric matrices. We relate to them functions for memory management, I/O operations, and generic operations when it is possible, for example to transpose or invert a MATRIX. In this way we obtain three objects representing the first layer of the $PIMA(GE)^2$ object hierarchy; they already include, as methods, some of the objects derived by the library classification and part of the hidden library code level.

3.2. The image processing objects

The core of the $PIMA(GE)^2$ server is a set of five objects, obtained by grouping together the library objects, according to the rules already mentioned:

1. **Point Operations** Operations that take in input only one IMAGE object, i.e. square root, absolute value, sine, or a collective value, i.e. the maximum, minimum pixel value of an IMAGE;

2. **Image Arithmetic** Operations that take in input two IMAGE objects, i.e. addition, product, etc;

3. **Geometric Operations** Operations that take in input one IMAGE object and one MATRIX object, i.e. rotation, translation and scaling;

4. **Convolution Operations** Operations that take in input one IMAGE object and one KERNEL object, for example involving a neighborhood of each pixel: percentile, median. or combined together by two binary functions: Gaussian convolution, erosion, dilation.

5. **Differential Operations** Operations that take in input one IMAGE and perform differential operators, i.e. Hessian, gradient, Laplacian.

4. The server internal structure

The main difficulty in the server design is due to the presence of parallelism in the computation. It imposes the management of two different environments: the CORBA framework (server side) and the MPI library (legacy code side). They are aimed to manage different kind of problems, therefore there is not a standard schema to allow their cooperation. In fact CORBA was born to develop mainly sequential applications, hence it does not support intrinsic compatibility with any kind of parallel environment. We decided to use a standard CORBA specification, TAO [17] , and look for a way allowing the execution of the parallel computation inside a CORBA object.
To achieve this goal we designated a process to perform specific tasks. We took advantage from the presence of the coordinator MPI process during the initialization and I/O phases; this process acts as the gateway that coordinates the two environments, allowing a cooperative computation. It has a dual position in the software architecture, in fact it will be at the same time a CORBA and a MPI process. On the CORBA side, it has to activate the ORB and to perform as a CORBA server; at the same time it is also one of the spawned MPI processes, and it has to manage the parallel computation. It performs as an MPI process that is able to interact with the CORBA environment, collecting the client requests, and managing the parallel computation in a properly way, i.e. invoking the services required from the clients with their proper parameters. This solution is possible because, acting as a CORBA server, the designate process knows the Client requirements and is able to communicate

them to the others MPI processes. In this way it coordinates the computation and gives back the information to the Clients. This solution does not require modifications to the legacy code, since it exploits the presence of the coordinator MPI process during the initialization and I/O phases. The behavior is presented in the Figure 2.

Another important feature of the $\mathrm{PIMA(GE)}^2$ server organization is concerned with data man-

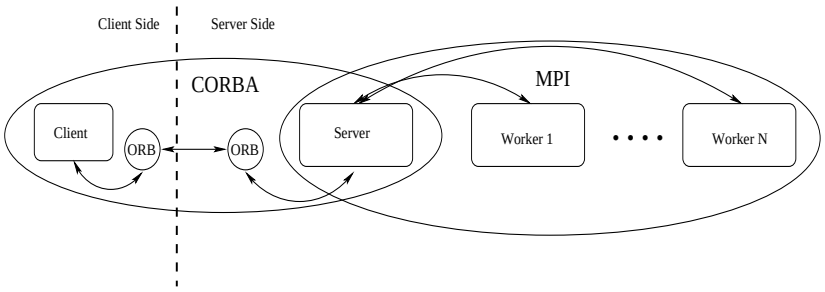


Figure 2. The CORBA-MPI interaction

agement. In order to minimize the overhead introduced in the wrapping phase, a good point is to improve the exploitation of the environment through the development of services oriented to the data transfer and smart strategies in the communication channel management. A considerable improvement in the computations is brought if data are managed by the most proper tools, as an example large size data transfer across a distributed environment can be more efficiently accomplished using optimized FTP protocols instead of the ORB channel. Providing optimized data transfer services also leads to a clear separation between movement, access, and processing of data. This point represents a key element in the exploitation of a distributed architecture.

At this aim we remark that an application calls a pipeline of library functions acting on the same data, with the same degree of parallelism. This remark permits to assume that a sequence of operations required by the client, may be performed entirely on the server on the set of images that are distributed once and kept on the parallel nodes till the end of the pipeline, thus avoiding useless movement of large data through the ORB channel. In order to force this type of interaction between the client and the server, we implemented the server in such a way that:

- The server executes a sequence of operations issued by the client as some kind of atomic action, in the sense that the parallel context and data distribution to parallel nodes remain unchanged during the whole sequence;
- The client-server interaction inside a sequence of image operations is limited to the exchange of symbolic identifier of images.

It is important to notice that these implementation strategy does not introduce heavy dependencies in the client code. The only aspect that the client has to take in consideration is that it has to interact with a CORBA server. An example is provided in the next Section.

5. Client-Server interaction example code

The Client code looks like that of a process that uses standard CORBA service and standard library routines; in fact the policy applied to allow MPI-CORBA compatibility is totally hidden to the

Client and managed by $\mathrm{PIMA(GE)}^2$ server. The optimization policy is embedded in the server code as well, and also the use of the symbolic identifiers is managed by the server and it is not perceived by the clients.
As it happens using a specific domain library, it is necessary to invoke the initialization and closure services. Nothing more is required. To perform a selected operation on an image, the Client has to relate an integer value to it, with a normal declaration; in the example code below it is the only one performed. The declared integer identifies its unique correspondent image during all the Client-Server session, and it corresponds to the symbolic identifier used to minimize the use of the ORB channel.
The example of Client code shows the transparence and easy of use of $\mathrm{PIMA(GE)}^2$ services.

Client code

```
int main(int argc, char* argv[]) {

 CORBA::ORB_var orb = CORBA::ORB_init (argc, argv,"");
 std::string ior; std::ifstream is(argv[1]); std::getline(is, ior);
 CORBA::Object_var obj = orb->string_to_object(ior.c_str());

//get the pimage object reference
 LIB::Operazioni_var pimage = LIB::Operazioni::_narrow(obj.in());

//declaration of the integer value used to identify an imag
 CORBA::Long origImg, resImg;

 pimage->InitLib();

//implicit association of the integer origImg to the image
//stored in the file "myimg"
 pimage->readImageFromFile_C("myimg",origImg);
 pimage->getdimension(origImg,imWidth, imHeight, imDepth);

//implicit association of the integer resImg to a new image
 pimage->createImage_C(imWidth, imHeight,imDepth, resImg);

 pimage->rotImg(IdImg, resImg, 60, LINEAR, 0, p);
 pimage->reflectImg(resImg,doX,doY);
 pimage->writeImageToFile_C(resImg,"myresult");
 pimage->deleteImage_C(origImg);
 pimage->deleteImage_C(resImg);
 pimage->CloseLib();

 orb->destroy();

 return 0; }
```

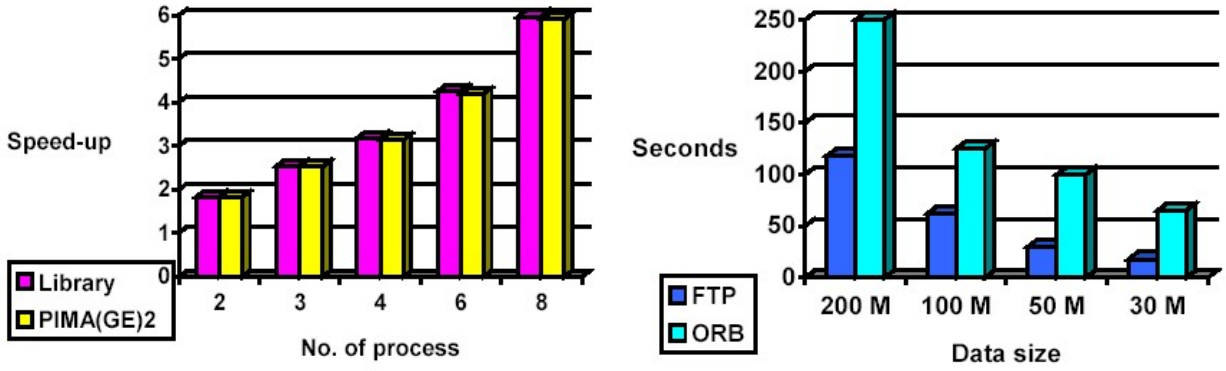
6. Experimental results

The experimental results we obtained show a reasonable overhead due to the presence of the CORBA framework. We carried out tests aimed to assess the performance of the $\mathrm{PIMA(GE)}^2$ server compared with the parallel library, and to evaluate the benefit obtained by minimizing the use of the ORB communication channel, during the invocation of a $\mathrm{PIMA(GE)}^2$ services sequence, and in order to transfer data from the client to the server. The application used for the tests is aimed to

the detection of linear structure in an image. In all the cases the parallel execution environment is a Linux cluster with eight nodes interconnected by a Gigabit switched Ethernet, and 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 a PC that is located on the same local area network, in order to be able to evaluate overheads that are due to our software infrastructure and limit other possible sources of overheads.

The tests assert that the CORBA set-up time is a fixed value (about 1-2 seconds in our experiments), and does not depend on the number of parallel MPI processes. When the interaction between the client and the server is going to grow, we have a performance degradation due to a more consistent use of the ORB communication channel; but applying the symbolic identifiers strategy the overhead is quite limited especially for compute intensive applications. Using symbolic identifiers is a strategy that leads to a fairly unvaried speed-up performance. Comparing the speed up of the application performed with the library functions and one obtained implementing the application with the $\mathrm{PIMA(GE)}^2$ services we obtained that the variation of the speed-up is around 1.3%; experimental results are plotted in Figure 3(a).

From the data transfer point of view, we prove that definitely better performance are obtained exploiting FTP protocols; in fact the execution time using the ORB channel is about twice of the one obtainend using the ftp protocol. This behavior is indipendent from the data size, according to the results presented in Figure 3(b).



(a) Speed-up comparisons

(b) Data transfer services comparisons

Figure 3. Graphics reporting the comparison between the speed-up of the library and $\mathrm{PIMA(GE)}^2$, the left one, and the comparison between the data services, the right one

7. Conclusion

An approach to high performance legacy code encapsulation in a grid-oriented environment is presented; we described a way to reuse an MPI-based parallel image processing library in distributed environment, using CORBA without modifications to the OMG standard. Also the legacy code had no modifications. To allow the cohesistence of CORBA and MPI environment, we exploit

the presence of a the coordinator MPI process during the initialization and I/O phases. In order to reduce the overhead introduced in the wrapping, the $\mathrm{PIMA(GE)}^2$ server architecture has been improved with more specific data transfer services. This step also allowed the use of an optimized ORB channel management strategy.

8. Acknowledgments

This work has been supported by MIUR programme L.449/97-00 High Performance Distributed Computing Platform, and by FIRB strategic project on Enabling Technologies for Information Society, Grid.it.

References

[1] R.F. Boisvert: Mathematical Software: Past, Present and Future. Computational Science, Mathematics, and Software, Purdue University Press, 2002.
[2] 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. 584-5891, 2005.
[3] The CORBA home page, http://www.corba.org/
[4] A. Dennis, C. Peréz and T. Priol: Portable parallel CORBA objects: an approach to combine parallel and distributed programming for Grid Computing. Euro-Par 2001
[5] A. Dennis, C. Peréz and T. Priol: PadicoTM: An open integration framework for communication middleware and runtimes. Future Generation Computer System, 19-4. May 2003
[6] I. Forster and C. Kesselman: The grid: blueprint for a new computing infrastructure, 2nd Edition. Morgan Kaufmann Publishers Inc. 2004.
[7] A. Galizia: Evaluation of optimization policies in the implementation of Parallel Libraries. Technical Report IMATI-CNR-Ge, N. 20/2004
[8] GrADS Project, Grid Application Development Software Projects, http://www.hipersoft.rice.edu/grads/
[9] J. Lebak, J. Kepner, H. Hoffmann and E. Rudtledge: Parallel VSILPL++: an open standard library for high-performance parallel signal processing. IEEE Proceedings, vol. 93-2. February 2005
[10] NetSolve Project, http://icl.cs.utk.edu/netsolve/
[11] OMG Official Website, http://omg.org
[12] Parallel Image Processing Toolkit Home Page, http://www.osl.iu.edu/research/pipt/
[13] C. Peréz, T. Priol and A. Ribes: A Parallel CORBA Objects for programming Computational Grids. Distributed Systems Online, 4-2. February 2003
[14] G. Ritter and J.Wilson: Handbook of Computer Vision Algorithms in Image Algebra, 2nd edition. CRC Press, Inc. 2001
[15] F. Seinstra, D. Koelma and J.M Geusebroek: A software architecture for user transparent parallel image processing. Parallel Computing, 28. 2002
[16] H.M. Sneed: Encapsulation of Legacy Software: A technique for reuse software components. Annals of Software Engineering, vol. 9. 2000
[17] Tao home page, http://www.cs.wustl.edu/schmidt/TAO.html
[18] N. Wang, D.C. Schmitdt and D. Levine: An overview of the CORBA component model. Addison-Wesley. 2000.