Scientific computing in the Latin America-Europe GISELA grid infrastructure

Sebastián García, Santiago Iturriaga, and Sergio Nesmachnow

Facultad de Ingeniería, Universidad de la República,
Montevideo, Uruguay
{sgarcia,siturria,sergion}@fing.edu.uy

Abstract. This work presents the application of parallel computing techniques to solve two scientific computing applications over the Latin America-Europe GISELA grid computing platform. The article describes two scientific computing applications –the semi-automatic processing of historical climate images and a software package for fluid dynamics– which usually require large computing times when applied to realistic scenarios. The proposal of applying parallel computing techniques over the GISELA grid infrastructure is formulated, and the implemented solutions are described. A preliminary experimental analysis is reported, presenting the estimated efficiency gains when using the grid infrastructure.

Keywords: grid computing, HPC, scientific computing

1 Introduction

Grid computing has become a new paradigm to deal with hard-to-solve problems, by cooperatively using a large set of computational resources distributed around the globe. Grid infrastructures provide great computing power, which is far larger than the available in single research institutions and universities, making possible to face more complex problems, increasing the accuracy of the solutions, and solving them in reasonable times. In our research context, the regional grid infrastructure has been developed by the *Grid Initiative for e-Science virtual communities in Europe and Latin America* (GISELA) project [5], which groups a large number of institutions from 19 countries on Latin America and Europe.

A specific area of application of the new grid computing infrastructures is scientific computing, a field where compute-intensive problems are very common. In this line of work, this article presents the application of parallel computing techniques to solve two scientific computing applications over the GISELA European-Latin American grid computing platform. Both scientific computing applications described in this work –the semi-automatic processing of historical climate images and a software package for fluid dynamics– have a great relevance for meteorological forecast, disaster control, and natural resources management. Both applications usually require large computing times when applied to model realistic scenarios. Thus, parallel computing techniques are needed in order to reduce the computing times, allowing to obtain accurate solution in reasonable times.

The main contribution of this article is to explain how this two scientific computing applications have been adapted to execute in a grid computing environment, and also to describe how to develop their distributed implementations over the European-Latin American grid infrastructure built by the GISELA project. A preliminary experimental analysis is also reported, and the estimated efficiency gains when using the grid infrastructure are presented.

The rest of the article is organized as follows. Section 2 briefly introduces the grid computing paradigm, and Section 3 describes the main components of the European-Latin American grid infrastructure built by the GISELA project. After that, Section 4 presents the two scientific computing applications parallelized in this work: Digi-clima and caffa3d.MB. The main details about the implementations of Digi-clima and caffa3d.MB over the grid infrastructure are provided in Section 5. A preliminary experimental analysis and the projected efficiency gains are reported in Section 6. Finally, Section 7 summarizes the conclusions of the research and formulates the main lines for future work.

2 Grid computing

In the last twenty years, distributed computing environments have been successfully employed to solve complex problems. The size and the computing power of distributed computing environments have significantly improved, mainly due to the fast increase of the processing power of low-cost computers and the rapid development of high-speed networking technologies. Nowadays, a common platform for distributed computing usually comprises a heterogeneous collection of computers able to work cooperatively for solving complex problems. At a higher level of abstraction, the expression *grid computing* has become popular to denote the set of distributed computing techniques that work over a large loosely-coupled virtual supercomputer, formed by combining together many heterogeneous components of different characteristics and computing power. This infrastructure has made it feasible to provide pervasive and cost-effective access to a collection of distributed computing resources for solving problems that demand large computing power [4].

Grid computing applies some key concepts from parallel computing: the workload distribution and the mechanisms used for the synchronizations/communications between distributed processes are crucial aspects to achieve high performance when using a loosely-coupled computing infrastructure. As well as the flow of data and instructions, which often define the type of parallelism applied in the grid; and portability issues, specific hardware requirements, and required run-time libraries and modules. These prerequisites have to be checked before implementing and/or executing a parallel application in a grid infrastructure. The large availability of computing resources in grid infrastructures also provides support to implement fault-tolerance mechanisms, such as checkpointing and restarting, multiple data repositories, etc. Due to all these features, grid computing has emerged as a powerful paradigm for solving complex problems in many application areas [1].

3 The GISELA grid infrastructure

GISELA [5] is a consortium of 19 partners (5 from Europe and 14 from Latin America), that aims at ensuring the long-term sustainability of the Latin American Grid Initiative (LGI) and provides virtual research communities (VRC) with e-Infrastructure and application-related services. The consortium partners are organized in Virtual Organizations, which share information, computing power, software, or other resources for collaborative problem-solving.

At May, 2011, GISELA has 1233 CPU and 63 TB of storage, in 21 centers. The pledged computational power to be integrated is of **2660** CPU, **105** TB of storage and **56** resource centers. GISELA is working the transition of the LGI sustainability to the Latin American Cooperation of Advanced Networks (CLARA).

GISELA uses the gLite middleware stack [8] as the default software platform. gLite was produced by the EGEE [2] project and it is used by a large number of scientific groups in the world. The services on gLite are organized in five groups: security, information and monitoring, job management, data storage, and helper. The main components are: User Interface (UI), which provides users with tools for job management; Computing Element (CE), which provides the computing power; Storage Element (SE), which provides data storage services; and the Workload Management System (WMS), for job scheduling.

Fig. 1 shows the main components of gLite. A CE is a set of computing resources (Worker Nodes - WNs), the computers where the jobs are run. The SE provides uniform access to data storage resources. All data in a SE is considered *read-only* and must be replaced on change. A *file catalog* service (LFC) is used to locate files or replicas in every SE in the grid. LFC maintains a mapping between the *logical file name* (LFN) and the physical *storage URL* of all of its replicas [7]. The WMS assigns the jobs to execute in the appropriate CE. Submitted jobs are described using the *Job Description Language* (JDL), which specifies which executable to run and its parameters, hardware and software requirements, etc.

The *ARDA Metadata Grid Application* (AMGA) is a metadata service on the grid that allows users to efficiently maintain large results and application data. A metadata catalog can be viewed as a simplified database of non-file related data, too small or too volatile to be stored in data files [6].

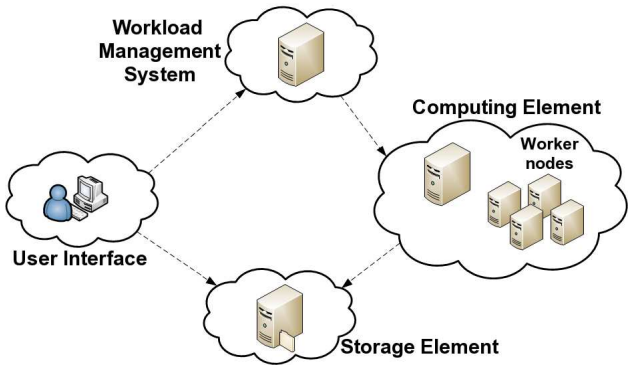

Fig. 1. Main components of gLite.

4 Two scientific computing applications

This section describes the scientific applications to parallelize in GISELA.

4.1 Digi-Clima

Digi-Clima is an Octave/Matlab application for the semi-automatic processing of historical graphical rain records. In our country, historical rain records are kept from the early 1900's, most of them in paper. All this data is of great value, but its use is limited due to its paper-format storage, and its preservation is in danger. Digi-clima aims at digitalizing the pluviographic records to preserve the data and to allow an easier access to it; these records are stored in graph paper bands containing the pluviometer output for a certain time period, which can be divided into intervals with each one looking like a continuous growing monotone function, Fig. 2 shows three scanned pluviometer output bands. More than 20000 data bands are available from the last 30 years of the national flooding and the stormwater management systems. Digitalizing one band takes about ten minutes, thus, parallel computing is critical to reduce the total processing time.

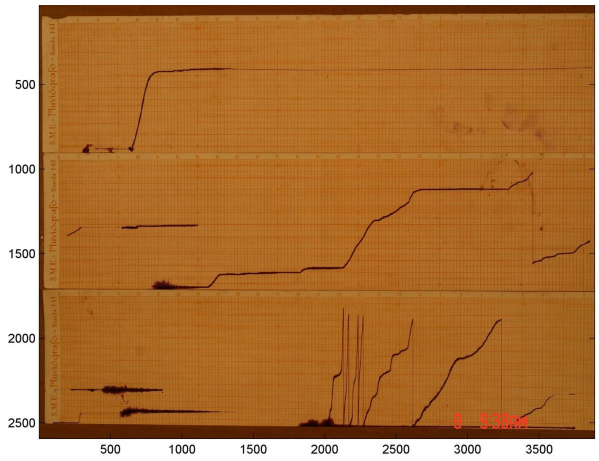

Fig. 2. Three scanned pluviometer output bands.

General program description. Digi-clima performs the following tasks: (1) *color separation*: separates the rain traces from the background and other information by color tagging, getting a black and white image which only contains active pixels corresponding to rain record traces; (2) *individual band identification*: the frame information layer is analyzed to separate and scale the individual bands; (3) *trace identification*: for each band, identifies traces of rain records as continuous lines; (4) *trace analysis*: analyzes each trace to obtain its $x(t)$ footprint in pixel scaling, by using a simple median estimator in Octave, or by using a spline fitting in Matlab; (5) *trace grouping*: orders the separate traces in each band, since these records should be monotonic. (6) *rain intensity computing*: obtains the rain intensity data from the discrete time derivative of the grouped traces.

Digi-Clima can be executed as: *interpreted Octave*, requires Octave 3.x and ImageMagick (both open source); *interpreted Matlab*, requires Matlab 7.x or higher with the curve fitting toolbox (licensed product); and *compiled Matlab*, without extra library requirements, but the CE must match the compiled binary architecture and the Matlab Compiler Runtime (MCR) must be available. The MCR is provided with MATLAB Compiler and can be deployed royalty-free.

The disk space required is about 3 MB per image. The required RAM depends on the records and the size of the image (usually, less than 1GB).

Parallel model. The gridification of Digi-Clima is based on *data parallelism*, which splits the whole data domain into smaller subdomains, and executes a single algorithm in parallel in each of them, as it is presented in Fig. 3.

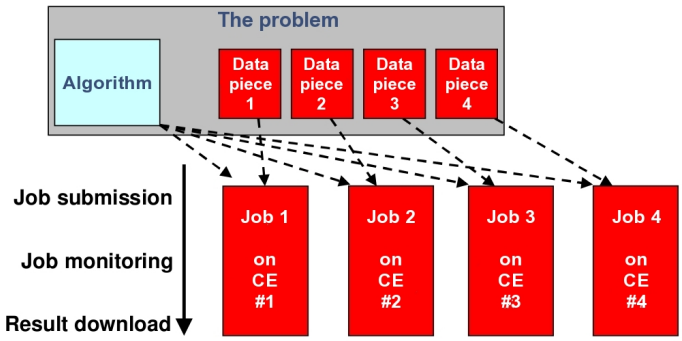

Fig. 3. Data parallelism model.

Applying data parallelism to Digi-clima is straightforward, because processing each image is independent from others. Also, as the final job output consists only of the union of each parallel process output, no post-processing is needed.

A master/slave parallel model was applied to Digi-clima (see Fig. 4). The master process launches and assigns the work to several slave processes that executes Digi-clima on a set of images. Once a slave finishes its work, the master process collects the output data to include it in the final job output.

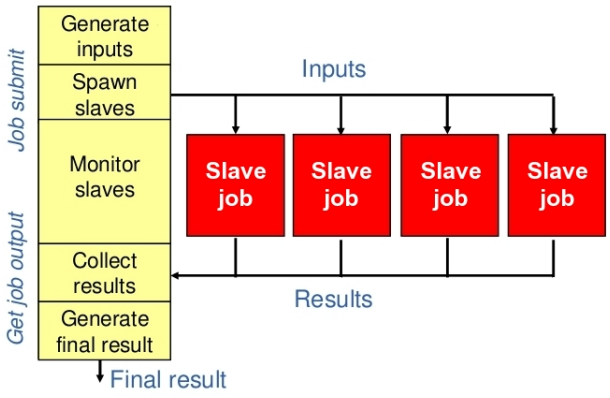

Fig. 4. Master/slave parallel model in Digi-clima.

4.2 caffa3d: 3D Navier-Stokes solver

caffa3d.MB implements a fully implicit finite volume method to solve the 3D Navier-Stokes equations in complex geometry, extending the 2D caffa.f by Ferziger and Peric [3]. The next subsections introduces the mathematical model and the solver in caffa3d.MB, and the proposal of applying parallel computing techniques in the exploration of the parameters space of the model.

General program description. The mathematical model in caffa3d.MB considers the mass balance equations and mass momentum equations for a Newtonian incompressible flow in a gravitational field, using the Boussinesq approximation to model the floating effects induced by small density variations due to temperature. The solver in caffa3d.MB includes block-structured, non-orthogonal, body fitted, collocated meshes for the spatial discretization and several other features to deal with complex geometries. For the time discretization, fully implicit two-level first order (implicit backward Euler) and three-level second order schemes are available. caffa3d.MB is mainly implemented in FORTRAN 77, including some FORTRAN 95/90 extensions.

Performance analysis. caffa3d.MB usually involves intense computing, due to three main groups of compute-intensive tasks: i) routines for updating the coefficient matrix for each equation (momentum, mass balance, etc); ii) routines for computing the gradients of each field through Gauss theorem; and iii) solving the heptadiagonal systems of equations using the ILU SIP solver [9]. Routines in i) and ii) imply visiting each cell interface in an ordered fashion (East faces Loop, North faces Loop, Top faces Loop) and computing the flux contributions to either the coefficient matrices or the gradients. Routines in iii) mainly involve backward and forward substitutions to solve tridiagonal systems of equations. In fact, the ILU SIP solver is the most time consuming routine in the code, requiring up to 30% of the total computing time.

Software and hardware requirements. caffa3d.MB can be compiled with GNU gFORTRAN, Intel ifort, and Portland Group gf90. Only standard libraries are required for the compilation and the execution. RAM requirements vary according to the mesh size, usually in the range of 0.5 to 6 GB.

Exploration of the caffa3d.MB parameter space. In engineering practice is often not enough to run a single simulation with a specific set of parameters values. The user usually needs to scan a large phase space -defined by a range of values- for a given number of parameters. For example, a given flow problem might exhibit critical behaviors at different values of the Reynolds number (RN), and the user might be interested in simulations with a given range, incrementing its value in steps. To perform this experimental analysis, a large number of independent executions are needed. Afterward, the user might want to refine some interesting region of the parameter space at smaller intervals of RN values. Thus, an efficient way of organizing and distributing the parameters exploration is a useful tool in practice. This is the basis of the specific proposal to implement using parallel computing techniques in a grid environment.

5 Implementation

This section presents the details of the distributed implementations of the Digi-clima image processing and the caffa3d.MB parameter exploration proposed to execute in the grid infrastructure.

5.1 Digi-Clima

The main issues to solve when executing Digi-Clima over a grid infrastructure are: i) to distribute the images; ii) to avoid processing a single image at the same time by two Digi-Clima instances; and iii) to retrieve the results and know to which image they belong.

The time to set up the CE workspace to run Digi-clima (i.e. at least download Matlab Compiler Runtime, which size is about 200MB.) is not negligible, and a large number of images are to be processed, the proposed implementation uses *pilot jobs*. Once submitted and assigned to a CE, the pilot job runs a loop, executing Digi-clima on each iteration to process multiple images without the delay due to job submission and CE setup. Grid file services are used to distribute the images and to get the results. A metadata managing service named AMGA is used for accounting on the images metadata as processing status and the mapping between the original image file names and their corresponding results.

The image uploading script. The image uploading script runs on the image directory. It creates the Digi-clima directory on AMGA and adds the attributes for image accounting information: the image *id* on the SE, the image *status*, the *original_name* of the image file, the timestamp of the *last_update*, and the *job_identifier* of the job which last updated the entry. After that, the `lcg-cr` command is used to upload the images. A sketch of the script is presented in Fig. 5 to show how the mentioned interaction with AMGA and storage services works.

```
1  mdcli createdir /schooldir/valparaiso/Digi-clima/images
2  mdcli "addattr $wPath/images id_in_se varchar(200)"
3  mdcli "addattr $wPath/images status varchar(1)"
4  mdcli "addattr $wPath/images original_name varchar(200)"
5  mdcli "addattr $wPath/images last_update int"
6  mdcli "addattr $wPath/images job_identifier varchar(200)"
7  image_count=$1
8  for image_name in `ls *.JPG`; do
9    image_count=$(expr $image_count + 1)
10   lcg-cr --verbose --vo prod.vo.eu-eela.eu \
11     -l /grid/prod.vo.eu-eela.eu/Digi-clima/images/"$image_count" \
12     -d se01-tic.ciemat.es $image_name
```

Fig. 5. Shell script to upload images and initialize AMGA.

Pilot job. The pilot job has two components: the JDL file to submit the job to the WMS, and the loop script that runs on a CE, launching on each step the script to set up and run Digi-clima for each image. Fig. 6 presents the execution sequence of the pilot job.

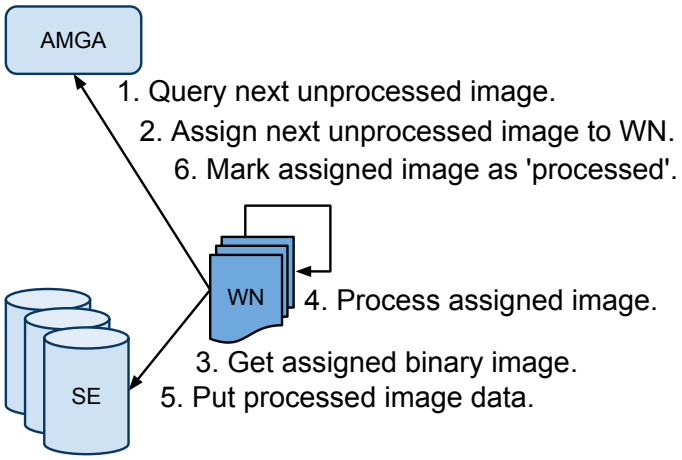

Fig. 6. Pilot job sequence diagram.

Pilot job descriptor . Fig. 7 presents the JDL file used for the submission of the pilot job. The shell script is run with the pilot job script as parameter using the given names for standard error and output files. Then, using the `InputSandbox` rule the pilot job descriptor specifies the WMS to send the scripts needed within the pilot job to the CE. After that, it is indicated to retry three times if the execution fails and at last, through the `Requirements`, the pilot job descriptor indicates that the job demands `x86_64` architecture to run.

```
1 [ Executable = "/bin/sh";
2   Arguments = "Digi-clima.pilot.sh";
3   StdError = "stderr.err";
4   StdOutput = "stdout.out";
5   InputSandbox={"Digi-clima.sh","pilot.sh","mdclient.config"};
6   OutputSandbox = {"stderr.err", "stdout.out"};
7   RetryCount = 3;
8   Requirements = (other.GlueHostArchitecturePlatformType == "x86_64");]
```

Fig. 7. Pilot job descriptor.

Pilot job shell script. First, the pilot job shell script sets up the environment to run `LCG` commands. After that, it locates the best replicas to download the files needed to run Digi-clima, specially the Matlab Compiler Runtime which is about 200 MB, and thus the main reason for using pilot jobs due to the download time. This is done by using the `lcg-lr` command to list the replicas for a given logical name, and then looking for a match with the default SE for the CE within its output. If a match is found, then the download is done from that SE, otherwise it tries from a known SE and if this operation fails too, it tries using the logical file name. When the runtime, the code, and the libraries are already downloaded, the pilot job shell script executes a loop running `Digi-clima.sh` for each image, uploading the result to the SE, until there are no images left (see a sketch of the pilot job shell script in Fig. 8).

```
export vo=prod.vo.eu-eela.eu # Set enviroment for LFC and LCG commands
export LCG_LOCATION=/opt/grid/ui/lcg
export LCG_GFAL_INFOSYS=bdii-eela.ceta-ciemat.es:2170
for each file do # Download needed files from best replica
  SEL_REPL= file_uri
  # Try to get best replica
  REPL=$(lcg-lr file_uri | grep "${$VO_PROD_VO_EU_EELA_EU_DEFAULT_SE}")
  if [ "$REPL" != "" ]; then # found replica at DEFAULT_SE -> use it.
    SEL_REPL=$REPL
  else # Try to use "se01-tic.ciemat.es".
    REPL=$(lcg-lr file_uri | grep "se01-tic.ciemat.es")
    if [ "$REPL" != "" ]; then # se01-tic.ciemat.es fails too, try LFN.
      SEL_REPL=$REPL
  lcg-cp --checksum --checksum-type md5 --verbose $SEL_REPL file_name
END_EXECUTION=0 # Run Digi-clima
while [ $END_EXECUTION -eq 0 ] ; do
  /bin/sh ${JOB_DIRECTORY}/Digi-clima.sh ${JOB_DIRECTORY}
  if [ $? == $ERROR_ON_ASSIGN -o $EXIT_STATUS == $OK ] ; then
    END_EXECUTION= 0 # Image processed by other, continue with next.
  else
    END_EXECUTION= 1 # Error in execution, end pilot job.
```

Fig. 8. Pilot job shell script.

Digi-clima.sh. The Digi-clima.sh shell script starts by checking AMGA for an unprocessed image, and selecting one of them. After that, it tries to update the status of the selected image to "being processed", by asserting the image status to be unprocessed. This way using the AMGA service to prevent concurrent instances from selecting the same image to process. If this update fails, it means that another job did the update in the time between the select and the update, so the job fails returning the corresponding non fatal error, so the pilot job can continue launching Digi-clima instances. If the update succeeds, the Digi-clima.sh shell script retrieves the image from the nearest SE as explained in 5.1, and launches the Matlab script for that image. After the image processing is done, the Digi-clima.sh shell script uploads the results to a SE and sets the image as processed. A sketch of the Digi-clima.sh shell script is presented in Fig. 9.

```
query_result=$(mdcli -c mdclient.config "SELECT FILE FROM $path/images
    WHERE status = '$AVAILABLE' LIMIT 1") # Get an unprocessed image.
if [ "$query_result" != "" ] ; then
  mdcli -c mdclient.config "updateattr ${path}/images/${query_result}
      status $ASSIGNED last_update $(date +%s) last_update_job \"$JOB_ID
      \" 'status=$AVAILABLE'" # Set image status as assigned
  if [ $? != 0 ] ; then # Rare case, other job got the image first.
    exit $ERROR_ON_ASSIGN
  # Find best replica and download image (as in pilot job shell script)
  lcg-cp --verbose --checksum --checksum-type md5 $BEST_REPLICA ${
      image_local_path}
  ./digi-clima $image_local_path # Process image and upload results
  lcg-cr --verbose --checksum --checksum-type md5 --vo prod.vo.eu-eela.
      eu -l ${path_LFN}/results/$query_result -d $DEFAULT_SE $RESULT
  mdcli -c mdclient.config "updateattr ${path}/images/${query_result}
      status $PROCESSED last_update $(date +%s) last_update_job \"
      $JOB_ID\" 'status=$ASSIGNED'" # Set image status as processed
else  # No more images to process, end.
  exit $END
```

Fig. 9. Digi clima launcher shell script (Digi-clima.sh).

5.2 caffa3d.MB: exploration of parameters space

The parameter exploration of caffa3d.MB in a grid infrastructure is conceived as a master/slave parallel program. The master process controls the search by performing the domain decomposition, assigning each slave the parameters values to execute caffa3d.MB. The master also decides when to refine a promising region of the parameter space, depending on the results obtained for any single execution of caffa3d.MB performed by the slaves.

The parametric exploration could be implemented by using *parametric jobs*. However, that is not the best choice to assure a correct load balancing when executing on heterogeneous environments, because the output of the whole parameter exploration can only be retreived after *all* jobs finish its execution. In order to improve the load balancing by allowing a simultaneous exploration of different regions of the parameter space using different parameter ranges, a *dynamic model* is proposed for the distributed application. In the dynamic model, the slave processes execute caffa3d.MB on demand, with the parameter values sent by the master. Load balancing strategies (e.g. by adjusting the parameter ranges to explore by each slave) are applied by the master to deal with different simulation times for each of the parameter values explored. The proposed model is well suited for a grid system, since the slave processes run independent (i.e non-communicating) tasks.

In the first implementation of proposed application reported in this work, the parameter to study is the Reynolds number. The master-slave grid application for the parameter exploration of caffa3d.MB is presented in Fig. 10.

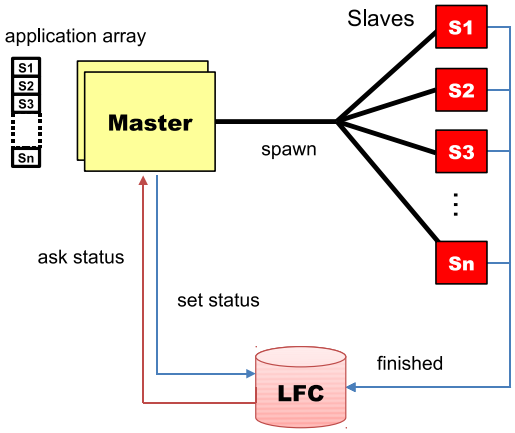

Fig. 10. Master-slave grid application for caffa3d.MB parametrization.

The master process is implemented as the shell script presented in Fig. 11.

```
INI=$1, STEP=$2, END=$3
par = $INI; apps = ( )
while par <= END # Submit the jobs (par in INI:STEP:END)
  apps = add (${apps[@]} $par) # Add application to array
  sed "s/PAR/$par/g" caffa.jdl.TEMPLATE > caffa.jdl.$par # Create jdl
  mkdir output_d.$par # Create output dir
  echo "" > status.$par # Create local file (status)
  lcg-cr -v --vo prod.vo.eu-eela.eu file:status.$par -l lfn:/grid/prod.vo.
      eu-eela.eu/caffa/status.$par -d se01-tic.ciemat.es # Register status
      LFC
  lfc-setcomment /grid/prod.vo.eu-eela.eu/caffa/status.$par 0 # Set
      status
  glite-wms-job-submit -o id_caffa.$par -a caffa.jdl.$par # Submit the job
  if [ $? == 0 ]
    echo "error $? in submit job $par"; exit 2
  par=`expr "$par" + "$STEP" `
while [ length(apps[@]) -gt 0 ]; do
  for i in ${!apps[*]}; do # Check the job status and process results
    if [ `lfc-ls --comment /grid/prod.vo.eu-eela.eu/caffa/status.$apps[$i]
        | awk '{ print $2 }'` -eq 1 ]; then
      unset apps[$i] # Application finished, delete from the array
      # Process results
      glite-wms-job-output --dir ./output_d.$apps[$i] -i id_caffa.$apps[$i]
      # Check if a refinement is required (i.e. if max_w>0.01)
      process_output ./output_d.\$apps[$i]/*/caffa*out
      if [ result -eq 1 ]; then # Execute the refinements
        NEW_STEP=`expr "$STEP" / 10 `
        INI=`expr "$apps[$i]" + "$NEW_STEP" `
        END=`expr $apps[$i] + 10 \* $NEW_STEP - 10`
        ./master.sh ${INI} ${NEW_STEP} ${END} &
  if [ length(apps[@]) -gt 0 ]; then
    wait 100 # Wait 100 seconds
```

Fig. 11. Master for caffa3D.MB parametrization.

The master spawns a given number of slave processes (which explores the parameters values from INI to END, with step STEP), and keeps an identification for each slave process created in the application array. After that, a JDL file with the specification of the job is created, using the template in Fig. 12. Then, the output directory and the status file are created. The status is registered in LFC, the 'unfinished' value is set to it, and the job is submitted. After spawning all the slave processes, the master iteratively checks for the status of each process. When a slave finishes the assigned task, the master gets the results of the execution (using `glite-wms-job-output`), and then it determines is a refinement is required. In this case, a recursive invocation is performed, after computing the new values for the arguments INI, FIN, and STEP.

```
Type = "Job";
JobType = "Normal";
Executable = "script_caffa.sh";
StdOutput = "caffa.out";
StdError = "caffa.err";
InputSandbox = {"script_caffa.sh","tar_del_caffa.tar"};
OutputSandbox = {"caffa.err","caffa.out"};
Arguments = "cavc41 PAR";
Requirements = (other.GlueHostArchitecturePlatformType == "x86_64")
```

Fig. 12. Template for caffa3D.MB JDL file.

The script that executes the caffa3D.MB in each slave is shown in Fig. 13.

```
export vo=prod.vo.eu-eela.eu
export LFC_HOST=lfc.eela.ufrj.br
export LCG_GFAL_INFOSYS=bdii.eela.ufrj.br:2170,bdii-eela.ceta-ciemat.es
    :2170
export LCG_LOCATION=/opt/grid/ui/lcg
tar -xvf tar_caffa.tar > /dev/null # Extract files
for i in `ls cav*gin`; do # Generate meshes
  echo $i | cut -d "." -f 1 > ./temp_grid
  ./grid3d.MB.9.0021.lnx < ./temp_grid > /dev/null
echo $1 > ./problem_name # Generate blocks
./block3d.MB.8.4005.lnx < ./problem_name > /dev/null
./caffa3d.MB.8.5002.lnx $2 # Execute caffa3d.MB
tar -cvf output$2.tar $1.out > /dev/null # Pack output
lfc-setcomment /grid/prod.vo.eu-eela.eu/caffa/status.$2 1 # Set status
lcg-cr -d $VO_PROD_VO_EU_EELA_EU_DEFAULT_SE -l lfn:/grid/prod.vo.eu-eela.eu
    /caffa/output$2.tar --vo prod.vo.eu-eela.eu file:$PWD/output$2.tar
```

Fig. 13. Script for executing caffa3d.MB.

6 Experimental analysis

Digi-Clima and caffa3d.MB were gridified during the *CHAIN/GISELA/EPIKH School for Application Porting* held in Valparaiso, Chile, 2010. The development and grid execution environments were provided as part of the school.

6.1 Digi-clima

A lightweight version of Digi-clima (using a simplified image processing algorithm) was used in the experimental analysis to process a test bank including 150 images (nearly 500 MB). The experimental analysis was performed on the resources provided by CIEMAT-TIC (90 CPUs and 35 TB of storage). Digi-Clima is a data intensive application, so it requires a fast access to the data to be processed. The application was stored and executed in the CIEMAT-TIC resource centre, thus locally accessing the data. This is not a limitation for executing in a distributed environment: gLite supports replicating data in multiple storage elements, guaranteeing an efficient access regardless of the resource centre used as long as adequate replicas are created.

Two test cases using 4 and 20 pilot jobs were used for the gridified lightweight Digi-clima. In the experimental evaluation, the pilot jobs waited 9 minutes (average) on the job queue before beginning execution. Furthermore, although the images are stored locally in the computing resource centre, the data download and result upload of each image takes 11 seconds on average. Table 1 summarizes the efficiency analysis of the lightweight Digi-clima. An estimation of the required execution time for the full Digi-Clima application (using the complete version of the image processing algorithm) is also included: the execution time should be reduced from about 20 hs. to 1.3 hs. using only 20 pilot jobs.

Scenario	**#images**	**pilot jobs**	**time**	**est. time**	**est. speedup**
Sequential	150	1	73.0 m.	20.0 hs.	1.00
Gridified_4	150	4	32.0 m.	5.3 hs.	3.77
Gridified_{20}	150	20	12.0 m.	1.3 hs.	15.05

Table 1. Digi-Clima application performance analysis.

The estimations show a promising nearly linear speedup. With this performance gain, the gridified Digi-clima should be able to process 20000 images in 135 hs. using 20 pilot jobs, compared to the 2665 hs. for the sequential time.

6.2 caffa3d.MB parameter exploration

The experimental evaluation used two test cases, considering the RN parameter in the intervals [50,1000] and [50,5000]. In both cases the flow behavior is initially studied for RN varying in steps of 50 units. Those regions identified as promising are explored considering smaller intervals for RN (varying in steps of 10 units)- The meshes and blocks requires below 1 GB RAM, and the sequential execution of caffa3d.MB for a given RN value takes about 20-30 minutes. The experiments were performed using the CIEMAT-TIC resource center (90 CPUs).

Case 1: RN in [50,1000]. This test performed 20 caffa3d.MB executions in the initial exploration, and 15 refinements in the 3 promising regions detected.

Thus, a sequential search take about 400 m. = 6.66 hs. (20 exec.×20 m.) for the initial exploration, and 300 m. = 5 hs. (15 exec.×20 m.) to perform the refined search, with a total execution time of about 700 m. = 11.66 hs.

The parallel execution in grid took a total time of 45 m. to perform the initial exploration, and 35 m. to perform the refinement, using 20 computing resources. The scheduling in the WMS/CE took in average about 10 m., and about 25-30 m. were spent in each execution of caffa3d.MB on the distributed WNs. The average overhead due to the asynchronous implementation, the time needed to download the program, and the access to LFC was about 3 m. The total execution time for the parallel search was 80 m. = 1.33 hs.

Case 2: RN in [50,5000]. This is a larger scenario that performed 100 executions in the initial exploration and 12 refinements. The estimated time for a sequential execution is 3200 m. = 53.33 hs. (160 exec×20 m.). The parallel execution took a total time of 210 m. = 3.5 hs. to perform using the 90 computer resources in CIEMAT-TIC.

Table 2 summarizes the execution times for the parameter exploration (sequential and parallel over the grid). Significantly high speedup values were obtained for the parallel application, specially for the largest test case, and the computational efficiency values were acceptable (the refinements must be performed *after* the initial search, thus the ideal computational efficiency is 0.5).

Scenario	**#WN**	**time (sequential)**	**time (grid)**	**speedup**
RN ∈ [50,1000]	20	11.66 hs.	1.33 hs.	8.75
RN ∈ [50,5000]	90	(estimated) 53.33 hs.	3.50 hs.	15.22

Table 2. Efficiency analysis for the caffa3d.MB parameter exploration.

7 Conclusions and future work

This work presented the application of parallel computing techniques to solve two scientific applications over the GISELA European-Latin American grid computing platform. The article described the digitalization of historical rain intensity data in the Digi-clima application and the numerical solver for computational fluid dynamics in the caffa3d.MB application.

The implementation details for the parallel Digi-clima and caffa3d.MB over the grid were presented, including a conceptual description of the parallel models used and the tools to implement the solution and execution over the grid.

A preliminary efficiency analysis was presented, demonstrating how the use of the grid infrastructure significantly help to reduce wall execution times required to solve these two complex problems.

Two main lines are formulated for future work: i) to further improve the efficiency analysis of the proposed parallel implementations of Digi-clima and caffa3d.MB, by solving even more complex scenarios and ii) to extend the proposed parallel grid techniques to other scientific computing applications. We are currently working on these topics now.

References

1. Buyya, R., Bubendorfer, K.: Market-Oriented Grid and Utility Computing. Wiley Publishing (2009)
2. EGEE: Enabling grids for e-science project, available at http://www.eu-egee.org/. Retrieved April 2011
3. Ferziger, J., Peric, M.: Computational Methods for Fluid Dynamics. Springer, Berlin (2002)
4. Foster, I., Kesselman, C.: The Grid: Blueprint for a Future Computing Infrastructure. Morgan Kaufmann Publishers (1998)
5. GISELA: Grid infrastructures for e-science virtual communities in europe and latin-america, available at http://www.gisela-grid.eu/. Retrieved April 2011
6. Koblitz, B., Santos, N., Pose, V.: The AMGA metadata service. Journal of Grid Computing 6, 61–76 (2008), http://dx.doi.org/10.1007/s10723-007-9084-6, 10.1007/s10723-007-9084-6
7. Kunszt, P., Badino, P., Frohner, A., McCance, G., Nienartowicz, K., Rocha, R., Rodrigues, D.: Data storage, access and catalogs in gLite. In: Local to Global Data Interoperability - Challenges and Technologies, 2005. pp. 166 – 170 (june 2005)
8. Laure, E., Fisher, S., Frohner, A., Grandi, C., Kunszt, P., Krenek, A., Mulmo, O., Pacini, F., Prelz, F., White, J., Barroso, M., Buncic, P., Hemmer, F., Meglio, A.D., Edlund, A.: Programming the grid with gLite. Computational Methods in Science and Technology 12(1), 33–45 (2006)
9. Stone, H.: Iterative solution of implicit approximations of multidimensional partial differential equations. SIAM Journal of Numerical Analysis 5, 530–538 (1968)