

# Scientific computing in the Latin America-Europe GISELA grid infrastructure

S. García, S. Iturriaga, S. Nesmachnow  
Universidad de la República, Uruguay

May 5, 2011

## 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.

## 1 Introduction

Grid computing has become a new paradigm for efficiently dealing with hard-to-solve problems, by cooperatively using a large set of computational resources distributed around the globe. Grid infrastructures provide a great computing power, far larger than the one available on 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 problems demanding large computing times 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 in our country. 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 these two scientific computing applications have been adapted to execute in a grid computing environment, and also to describe how to develop their parallel 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. Other important features are the flow of data and instructions, which often define the type of parallelism applied in the grid; and the portability issues, the specific hardware requirements, and the run-time libraries and modules needed to execution. 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 providing virtual research communities (VRC) with the 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 April, 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 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 the User Interface (UI), which provides the users with the commands required to manage the jobs; the Computing Element (CE), which provides the computing power; the Storage Element (SE), which provides the data storage services; and the Workload Management System (WMS), for the job scheduling. Figure 1 shows the main components of gLite.

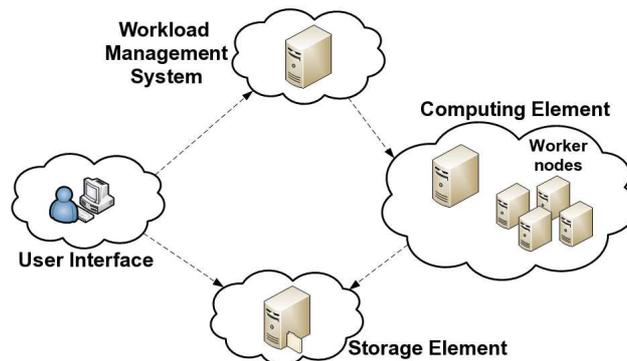


Figure 1: Main components of gLite.

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* (SURL) 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].

## 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. The records are stored in paper data bands, such as the one showed in Figure 2. 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.

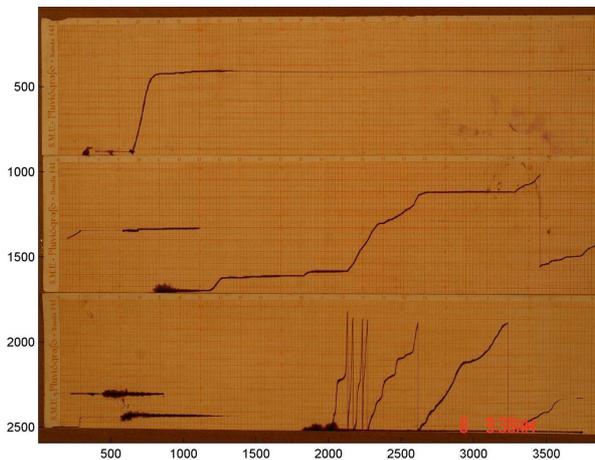


Figure 2: Original 3-band image example.

**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 the active pixels from the rain 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 the 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 requirements, but the CE must match the compiled binary architecture.

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 Figure 3.

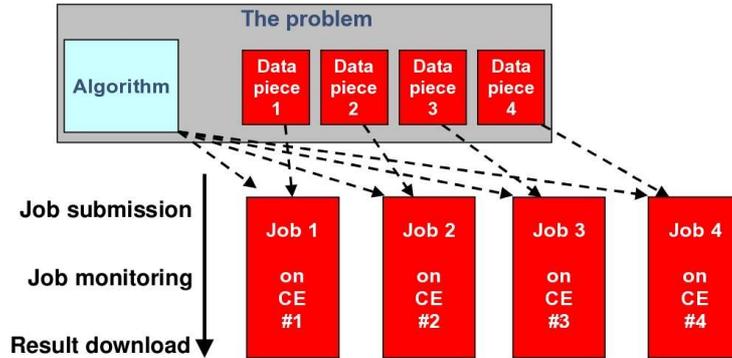


Figure 3: Data parallelism model.

Applying data parallelism to Digi-clima is straightforward, because the parallel processes are independent. 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 Figure 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.

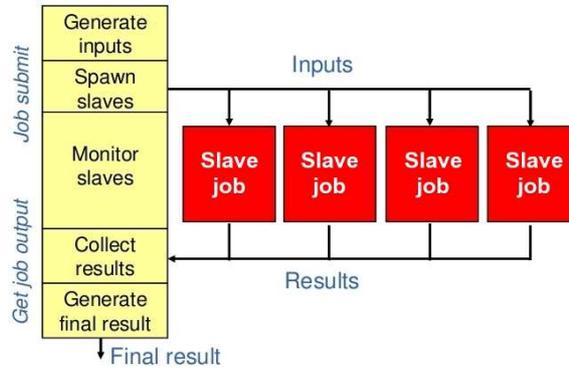


Figure 4: Master/slave parallel model in Digi-clima.

## 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 parallel method proposed to explore 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. 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, implicit two-level first order and three-level second order schemes are available. `caffa3d.MB` is implemented in FORTRAN 77, including some FORTRAN 95/90 extensions.

`caffa3d.MB` usually demands a large execution time, due to three main groups of time consuming tasks: i) updating the coefficient matrix for each equation (momentum, mass balance, etc); ii) computing the gradients of each field through Gauss theorem; iii) solving the systems of equations using the ILU SIP solver [9]. Routines in i) and ii) imply visiting each cell interface in an ordered fashion and computing the flux contributions. Routines in iii) mainly involve backward and forward substitutions to solve tridiagonal systems of equations. In fact, SIPSOL is the most time consuming routine in the code (up to 30% of the total computing time).

`caffa3d.MB` can be compiled with GNU gFORTRAN, Intel ifort, and Portland Group gf90. No libraries are required for the compilation or the execution. RAM requirements vary according to the mesh size, 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 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 parallel implementations of Digi-clima and `caffa3d.MB`.

### 5.1 Digi-Clima

The main issues to solve when executing Digi-Clima on the grid are: i) to distribute the images; ii) to avoid processing the same image 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 is not negligible, and a large number of images are to be processed, so *pilot jobs* are used. Once on 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. AMGA is used for accounting on the images metadata (status and name-results mapping).

**The image uploading script.** The image uploading script (Figure 5) 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* for the image file, the timestamp of the *last\_update*, and the *job\_identifier* for the last update. After that, the `lcg-cr` command is used to upload the images.

```

1 mdcli mkdir /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

```

Figure 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. Figure 6 presents the execution sequence of the pilot job.

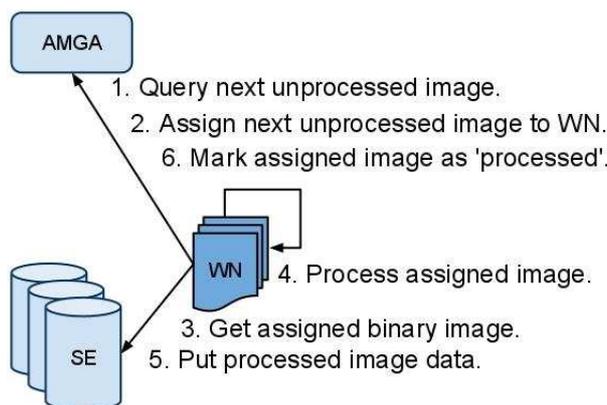


Figure 6: Pilot job sequence diagram.

*Pilot job descriptor.* Figure 7 presents the JDL file for the submission of the pilot job. The shell script is run with the pilot job script as parameter. The `InputSandbox` rule indicates the WMS to send the scripts needed within the pilot job to the CE, and the `Requirements` indicates that the job demands the `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");]

```

Figure 7: Pilot job descriptor.

*Pilot job bash script.* First, it locates the best replicas to download the files needed to run Digi-clima (specially the Matlab Compiler Runtime, which is about 200 MB, one of the main reasons for using pilot jobs). When the runtime, the code, and the libraries are ready, it executes a loop running `Digi-clima.sh` for each image, until there are no images left (see a sketch in Figure 8).

```

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

```

Figure 8: Pilot job bash script.

**Digi-clima.sh.** The `Digi-clima.sh` shell script checks AMGA for an unprocessed image, retrieves the image from the nearest SE, and launches the Matlab script. After the processing, it uploads the results to a SE and sets the image as processed (see a sketch in Figure 9).

```

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

```

Figure 9: Digi clima launcher bash script.

## 5.2 caffa3d.MB: exploration of parameters space

The grid parameter exploration of `caffa3d.MB` 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`, and also deciding when to refine a promising region of the parameter space, depending on the results obtained for any single execution.

The parametric exploration could be implemented using *parametric jobs*, but that is not the best choice to assure a correct load balancing when executing on heterogeneous environments. Then, a *dynamic model* is proposed for the application: the slave processes execute `caffa3d.MB` on demand, with the parameter values sent by the master. Load balancing strategies are applied by the master to deal with different simulation times. This model is well suited for a grid system, since the slave processes run independent non-communicating tasks.

In the proposed application, the parameter to study is the RN. Figure 10 describes the master-slave grid application for the parameter exploration of `caffa3d.MB`.

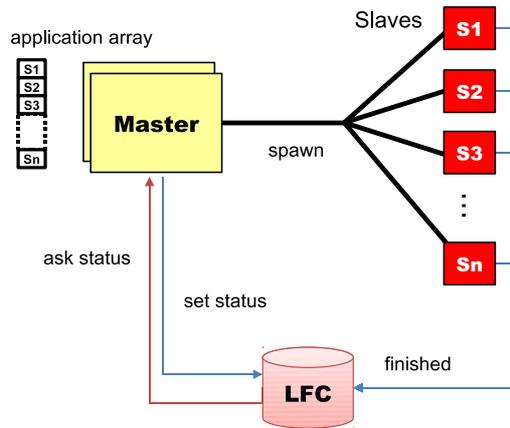


Figure 10: Master-slave grid application for `caffa3d.MB` parametrization.

The master process is implemented as the bash script presented in Figure 11.

```

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

```

Figure 11: Master for `caffa3D.MB` parametrization.

The master spawns several slave processes (from INI to END, with step STEP), and keeps an identification for each of them in the application array. After that, a JDL file with the specification of the job is created, using the template in Figure 12. Then, the output directory and the status file are created. The status is registered in LFC, the unfinished value is set, and the job is submitted. After spawning all the slave processes, the master checks for the status of each process. When a slave finishes its execution, the master process the results and it determines if a refinement is required. In this case, a recursive invocation is performed, after computing the new values for INI, FIN, and STEP.

```

1 Type = "Job";
2 JobType = "Normal";
3 Executable = "script_caffa.sh";
4 StdOutput = "caffa.out";
5 StdError = "caffa.err";
6 InputSandbox = {"script_caffa.sh","tar_del_caffa.tar"};
7 OutputSandbox = {"caffa.err","caffa.out"};
8 Arguments = "cavc41 PAR";
9 Requirements = (other.GlueHostArchitecturePlatformType == "x86_64")

```

Figure 12: Template for caffa3D.MB JDL file.

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

```

1 export vo=prod.vo.eu-eela.eu
2 export LFC_HOST=lfc.eela.ufrj.br
3 export LCG_GFAL_INFOSYS=bdii.eela.ufrj.br:2170,bdii-eela.ceta-ciemat.es:2170
4 export LCG_LOCATION=/opt/grid/ui/lcg
5 tar -xvf tar_caffa.tar > /dev/null # Extract files
6 for i in `ls cav*gin`; do # Generate meshes
7     echo $i | cut -d "." -f 1 > ./temp_grid
8     ./grid3d.MB.9.0021.lnx < ./temp_grid > /dev/null
9     echo $i > ./problem_name # Generate blocks
10    ./block3d.MB.8.4005.lnx < ./problem_name > /dev/null
11    ./caffa3d.MB.8.5002.lnx $2 # Execute caffa3d.MB
12    tar -cvf output$2.tar $1.out > /dev/null # Pack output
13    lfc -setcomment /grid/prod.vo.eu-eela.eu/caffa/status.$2 1 # Set status
14    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

```

Figure 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 processing data. 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 <sub>4</sub>	150	4	32.0 m.	5.3 hs.	3.77
Gridified <sub>20</sub>	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 $\in$ [50,1000]	20	11.66 hs.	1.33 hs.	8.75
RN $\in$ [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 the 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] R. Buyya and K. Bubendorfer. *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] J. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics*. Springer, Berlin, 2002.
- [4] I. Foster and C. Kesselman. *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] B. Koblitz, N. Santos, and V. Pose. The AMGA metadata service. *Journal of Grid Computing*, 6:61–76, 2008. 10.1007/s10723-007-9084-6.
- [7] P. Kunszt, P. Badino, A. Frohner, G. McCance, K. Nienartowicz, R. Rocha, and D. Rodrigues. Data storage, access and catalogs in gLite. In *Local to Global Data Interoperability - Challenges and Technologies, 2005*, pages 166 – 170, june 2005.
- [8] E. Laure, S. Fisher, A. Frohner, C. Grandi, P. Kunszt, A. Krenek, O. Mulmo, F. Pacini, F. Prelz, J. White, M. Barroso, P. Buncic, F. Hemmer, A. Di Meglio, and A. Edlund. Programming the grid with gLite. *Computational Methods in Science and Technology*, 12(1):33–45, 2006.
- [9] H. Stone. Iterative solution of implicit approximations of multidimensional partial differential equations. *SIAM Journal of Numerical Analysis*, 5:530–538, 1968.