Monitoring and Migration of a PETSc-based Parallel Application for Medical Imaging in a Grid computing PSE*

A. Murli[1], V. Boccia[1], L. Carracciuolo[2], L. D'Amore[1], G. Laccetti[1], and M. Lapegna[1]

[1] University of Naples Federico II, Naples, Complesso Universitario M. S. Angelo, Via Cintia, Naples (almerico.murli, vania.boccia, luisa.damore, giuliano.laccetti, marco.lapegna))@dma.unina.it

[2] Institute of High Performance Computing and Networking of CNR, Naples, ITALY, luisa.carracciuolo@na.icar.cnr.it

1 Introduction

In last decades, imaging techniques became central to the diagnostic process providing the medical community with a fast growing amounts of information held in images. This implies developing computational tools which allow a reliable, robust and efficient processing of data and enhanced analysis. Moreover, clinicians may have the need to explore collaborative approaches and to exchange diagnostic information from available data. A medical experiment often involves not a single approach but a set of processings that should be sometimes executed concurrently.

Grid computing is becoming a cost effective emerging technology for high performance computing aggregating resources that cannot be available locally [16]. In particular, grid technologies are a promising tool to deal with current challenges in medical domains. On the other hand, employing a distributed infrastructure, where nodes may be geographically scattered all around the world and not dedicated to a specific application, is not without a price. The challenge of the grid computing paradigm derives mainly from the dynamic nature of resource requirements. In this context, in particular, reliability is a key issue and critical to the correct diagnosis.
Here we are concerned with improvements and enhancements of a medical imaging grid enabled infrastructure, named MedIGrid, oriented to the transparent use of resource-intensive applications for the management, processing and visualization of biomedical images [5, 8, 6]. MedIGrid has been designed so that users can schedule reconstruction jobs needed in tomographic nuclear imaging or the denoising of ultrasound images arising in 3D echocardiography.
In this paper we focus on the optimization of the software routines of MedIGrid for dynamically adapting to changes in the computational nodes. More

* This work has been partially supported by the 2004-2006 PRIN 200415818 Italian National Project and the 2000-2006 PON SCoPE Italian National Project.

precisely, we deal with the monitoring and the migration of a parallel algorithm based on the PETSc library [3] for denoising of 3D ultrasound images.
The paper is organized as follows: in Sec. 2 a brief description of MedIGrid infrastructure is presented, in Sec. 3 the images reconstruction application is introduced, in Sec. 4 the Performance Contract System and its implementation inside the PETSc parallel algorithm is discussed, finally in Sec. 5 conclusions and future work are presented.

2 MedIGrid infrastructure

The testbed we are presently using, is made of acquisition systems and storage resources located in Florence (Careggi Hospital) and Genoa (S. Martino Hospital), computational resources in Naples and Lecce, grid access points in Naples and Genoa. The client side allows to set up the input and to monitor the reconstruction process by means of a user-friendly graphical interface. More precisely, the computational servers are:

IA-64-1 : cluster of 60 nodes operated by the INFN (Istituto Nazionale Fisica Nucleare). Each node of the cluster is composed by two Itanium 2 processors running at 1.4 GHz and with 4 GB of main memory. The nodes are connected by a switch Quadrics QSNet II. The operating system is Red Hat Enterprise 3 Linux, equipped with the hp-mpi, PETSc 2.2.1 and Autopilot;

IA-64-2 : cluster located at the University of Lecce with the same features of `IA-64-1`;

UniPart1 : cluster of 8 nodes located at the University of Naples, Parthenope. Each node is an Intel Pentium 4 HT running at 3 GHz with a main memory of 512 MB. The operating system is Fedora Core 3 Linux, equipped with mpich 1.2.7, PETSc 2.2.1 and Autopilot;

UniPart2 : cluster of 25 nodes operated by the University of Naples, Parthenope. Each node is an Intel Pentium 4 HT running at 2.8 GHz with a main memory of 512 MB. The operating system is Fedora Core 3 Linux, equipped with mpich 1.2.6, PETSc 2.2.1 and Autopilot.

The clusters in Naples are connected by a 1 Gbits metropolitan area network, while Naples and Lecce are linked by a 155 Mbits wide area network; the Globus 4 middleware has been used to build the computational grid.

More recently, the system has been upgraded with several new, advanced features, including grid services, available through the User Portal; an application oriented brokering service, as part of the application manager, to enable dynamic discovery and allocation of computing resources, an xml based configuration model to set parameters related to the execution of the software [7].

The following section describes the parallel algorithm that we have developed, using PETSc, for denoising a sequence of 3D ultrasound images of the heart [9].

3 The PETSc-based parallel algorithm

A 3-D image is the function

$$u_0(x_1, x_2, x_3) : \ \Omega \longrightarrow \Re_0^+, \quad \Omega \subset \Re^3; \tag{1}$$

a 3-D sequence of images is the function:

$$u_0(x_1, x_2, x_3, \theta) : \ \Omega \times I \longrightarrow \Re_0^+, \quad \Omega \subset \Re^3, \tag{2}$$

where $I := [0, T]$ is the time interval during which the acquisition of the sequence has been performed. We consider the following equation describing the denoising of the 3-D sequence:

$$\frac{\partial u}{\partial t} = clt(u) \nabla \cdot (g(|\nabla u_\sigma|) \nabla u); \tag{3}$$

[11, 22]. Equations are accompanied with zero-Neumann boundary conditions in space, initial condition is given by (2); finally, we suppose periodic boundary conditions in time.
The function *clt(u)* is a scalar function representing a measure of coherence in time for the moving structures [21]; g = g(s) is a continuous function satisfying:

$$g(0) = 1, \ lim_{s \to \infty} g(s) = 0, \tag{4}$$

and $u_\sigma := G_\sigma * u$ is obtained convolving u with a 3-dimensional Gauss function of zero mean and variance equal to σ,

$$G_\sigma = \frac{1}{(2\sqrt{\pi\sigma})^3} e^{-|x|^2/4\sigma}. \tag{5}$$

In order to compute u_σ we have to solve the Heat equation:

$$\frac{\partial u}{\partial t} = \nabla \cdot (\nabla u) \tag{6}$$

in $[0, \sigma]$ with initial condition u_0. We now briefly describe a common numerical scheme for the discretization of (3). Details can be found in [9].
Let $N = n1 \times n2 \times n3$ be the dimension of the 3-D frame, *nscales* the number of scale steps that are performed and, finally, τ be the discrete scale step; let us consider a space-time sequence consisting of n_4 3-D frames of dimension N, and let $\theta := T/(n_4 - 1)$ be the discrete time step; we denote by u_j^i the *jth* frame in the *ith* scale step,

$$u_j^i(x_1, x_2, x_3) := u(i\tau, x_1, x_2, x_3, j\theta), \tag{7}$$

where

$$\begin{aligned} i &= 0, 1, \cdots, nscales - 1, \\ x_1 &= 0, 1, \cdots, n_1 - 1, \\ x_2 &= 0, 1, \cdots, n_2 - 1, \\ x_3 &= 0, 1, \cdots, n_3 - 1, \\ j &= 0, 1, \cdots, n_4 - 1. \end{aligned}$$

Numerical discretization has been performed by using a semi-implicit scheme in scale, that is the nonlinearities are treated using the previous scale step, then linearized, while the linear terms are handled implicitly. Semi-implicit discretizations of $u_j^i(x_1, x_2, x_3)$ is shown in Fig. 1.

```
$u(x,0) = u_0$
% loop over the scales
**for** $i = 1, nscales$ **do**
    % loop over the frames
    **for** $j = 0, m$ **do**
        solve $\frac{u_j^i - u_j^{i-1}}{\tau} =$
        $clt(u_j^{i-1})\nabla \cdot (g(|\nabla(u_j^{i-1})_\sigma|)\nabla u_j^i)$
    **endfor**
**endfor**
```

Fig. 1. semi-implicit scheme for the numerical solution of (3).

The semi-linear discrete equations that arise, i.e. :

$$\frac{u_j^i - u_j^{i-1}}{\tau} = clt(u_j^{i-1})\nabla \cdot (g(|\nabla(u_j^\sigma)|)\nabla u_j^i), \tag{8}$$

are discretized in space via finite volume method [18]; we solve (6) with a semi-implicit scheme in scale as well, that is:

$$\frac{u_j^\sigma - u_j^{i-1}}{\sigma} = \nabla \cdot (\nabla u_j^\sigma), \tag{9}$$

where finite volume discretization in space has been used as for the equation (8).

Two main computational kernels arise, that is, the solution at each scale step i and for each frame j, of the linear systems:

$$\mathbf{A}_{HE}\, \mathbf{u}_j^\sigma = \mathbf{b}_j^i, \tag{10}$$

$$\mathbf{A}_{ME}\, \mathbf{u}_j^i = \mathbf{b}_j^i, \tag{11}$$

with

$$\mathbf{A}_{HE}, \mathbf{A}_{ME} \in \Re^{N\times N}, \mathbf{b}_j^i \in \Re^N,$$

where (10) refers to the space discretization of (9) and (11) to that of (8).
In Fig. 2. a schematic description of the algorithms that we have implemented is shown. The matrix $\mathbf{A}_{HE}$ depends upon σ and the size of the space discretization grid, so it is built only once, while $\mathbf{A}_{ME}$ depends upon quantities that change their value both with the scale step and the frame. As a consequence, its entries have to be recomputed $m \times nscales$ times. Right-hand side $\mathbf{b}_j^i$ contains the values of $\mathbf{u}_j^{i-1}$, i.e. the frame at the previous scale.

```
build $\mathbf{A}_{HE}$
% loop over the scales
for $i = 1, nscales$ do
    % loop over the frames
    for $j = 0, m$ do
        1. Init $\mathbf{A}_{HE}\mathbf{u}_j^\sigma = \mathbf{b}_j^i$
        2. solve $\mathbf{A}_{HE}\mathbf{u}_j^\sigma = \mathbf{b}_j^i$
        3. build $\mathbf{A}_{ME}$
        4. solve $\mathbf{A}_{ME}\mathbf{u}_j^i = \mathbf{b}_j^i$
    endfor
endfor
```

Fig. 2. multiscale analysis of a sequence of 3-D frames: outline of the algorithm.

Both $\mathbf{A}_{HE}$ and $\mathbf{A}_{ME}$ are large, sparse and structured; more precisely, their non-zero elements are located along seven diagonals: the principal diagonal and the three upper and lower diagonals respectively. Since we use the same discretization scheme both for (9) and (8), $\mathbf{A}_{HE}$ and $\mathbf{A}_{ME}$ have the same symmetric sparsity pattern; finally, $\mathbf{A}_{HE}$ is symmetric with respect to its entries as well, while $\mathbf{A}_{ME}$ is not for the presence of function $clt(u)$ in equation (3).
$\mathbf{A}_{HE}$ and $\mathbf{A}_{ME}$ are positive definite M-matrices symmetric in their structure. $\mathbf{A}_{HE}$ is besides symmetric with respect to its entries too. The properties we mentioned motivate the effectiveness of two popular Krylov projection methods, the Conjugate Gradient (CG) and the General Minimal Residual (GMRES) [13]. Both CG and GMRES are provided by the PETSc library [3]. Parallel approach is domain decomposition-based, i.e., we distribute the image domain among processes. In particular, we choose the **Slice Partitioning**, that is, the image is partitioned along one single dimension. Let $\Omega \subset \Re^3$ be the image domain, as defined in (1) and (2); Ω is a rectangular domain, with n_1, n_2, n_3 be its three dimensions. We distribute the domain along the third dimension only: if p is the number of processes, each process id, $0 \leq id \leq p-1$, will have

n_{id} slices of the 3-D image (that is, $n_1 \times n_2 \times n_{id}$ voxels), where:

$$n_{id} = \begin{cases} n_3/p + 1 & id < (n_3/p) \\ n_3/p & otherwise. \end{cases} \tag{12}$$

Let (i, j, k) be the coordinates of a voxel V_l in the $n_1 \times n_2 \times n_3$ image; voxels are numbered in a row-major fashion on successive planes, so, since each voxel generates one equation of the linear system and l corresponds exactly to the number of the equation generated by V_l, it follows that the slice partitioning gives rise to a *row-block fashion* distribution of the system matrix, that is, blocks of contiguous rows are distributed among contiguous processes. Slice partitioning has been chosen because the row-block fashion distribution is the standard PETSc matrices decomposition, and redistribution before the solution of the linear systems, is avoided.

4 Monitoring and Migration of the algorithm

The software architecture of MedIGrid is composed of three main layers: *core services*, based on the Globus toolkit, *collective services*, including the Resource Broker (RB) and the Performance Contract System (PCS), and the *Application Manager* (AM) that collects the software units that deeply interact with the algorithms during their execution. Some previous works refer to the Performance Contract System and to the Resource Broker [10, 17]. Here we focus on the deployment of the PCS for steering the performance of the parallel algorithm as described in section 3 in Figure 2. To this aim, following [2, 4] the reference workflow of the AM can be sketched as follows: the AM invokes the Performance Modeler with input parameters and information related to the computational resource. The Performance Modeler provides an execution model of the algorithm. The execution model, the input parameters and the machine parameters are given as "contract" to a Contract Developer. If the contract is approved the AM provides to spawn the job on the given resource. The Contract Monitor, monitors the times taken by the application while the AM waits for the job to complete. The job can either complete or, in case of contract violation, suspend its execution. If the job has completed the AM exits. If the job is suspended, the AM collects new information given by the brokering service and by the Contract Developer and it starts the phase again. In this latter phase a migration of the application onto another available resource can occur in such a way that the performance contract is satisfied. Hence, the entire process consists of a Periodical rescue of the execution state (**recovery**); a Run-time check of the execution flow (**monitoring**); a Process resumption on alternative resource (**migration**).

The contract verification consists of comparing the execution time of the algorithm with the one stated in the Performance Contract itself. We consider, as expected performance, the execution time of a computational kernel of the algorithm. In particular, we consider the execution time needed for denoising frame 1 at scale 1.

To monitor the algorithm, the *Autopilot* library [20] is used. The algorithm is instrumented by means of *sensors* and *actuators* to enable it to adapt its flow according to the performance level.

The migration step is aimed to suspend and migrate the execution of the algorithm on another resource in such a way that the performance contract is satisfied. We enable the parallel algorithm for saving current state and for restarting on another resource. More precisely, as shown in Figure 2, the algorithm consists essentially of two nested loops: the outermost over scale i and the innermost over the frame j. Denoising a single frame at each scale is performed by four steps: the first (*init*) and the third (*build*) steps have a computational complexity that does not depend on i and j, whereas the computational cost of the second (the parallel CG) and the fourth (the parallel GMRES) depend on the number of iterations needed to reach the requested tolerance, hence it depends on i and j. Note that both CG's steps and GMRES's steps have a computational cost that depends only on the size of the frame.

In order to select a resource on the basis of its computational power it is a common way to run a benchmark by the Performance Modeler [1, 14]. As benchmark of the k-th computational nodes of the grid, we consider the execution time, $T^{(0)}(1,1)$, needed for denoising frame 1 at scale 1:

$$T^{(k)}(1,1) = T^{(k)}_{init} + T^{(k)}_{CG}(1,1) + T^{(k)}_{build} + T^{(k)}_{GMRES}(1,1) \tag{13}$$

where:

- $T^{(k)}_{init}$ is the execution time of step 1
- $T^{(k)}_{CG}(1,1)$ is the execution time of step 2 on frame 1 at scale 1
- $T^{(k)}_{build}$ is the execution time of step 3
- $T^{(k)}_{GMRES}(1,1)$ is the execution time of step 4 on frame 1 at scale 1

Starting from the benchmark on the first frame we can provide an estimate of the execution time of the algorithm on a generic frame. Let:

- $\Delta_{CG}(i,j)$ be the number of iterations of step 2 on the frame j at scale i
- $\Delta_{GMRES}(i,j)$ be the number of iterations of step 4 on the frame i at scale j

then

$$\frac{\Delta_{CG}(i,j)}{\Delta_{CG}(1,1)} T^{(k)}_{CG}(1,1) \quad \text{and} \quad \frac{\Delta_{GMRES}(i,j)}{\Delta_{GMRES}(1,1)} T^{(k)}_{GMRES}(1,1)$$

provide respectively an estimate of the execution time of CG and of $GMRES$ on frame i at scale j.

Using the benchmark given by (13), taking into account that both $T^{(k)}_{init}$ and $T^{(k)}_{build}$ do not depend on i and j, the expected execution time needed for denoising frame j at scale i on the k-th node, and used by the Performance Contract, is the following:

$$PC^{(k)}(i,j) = T^{(k)}_{init} + \frac{\Delta_{CG}(i,j)}{\Delta_{CG}(1,1)} T^{(k)}_{CG}(1,1) + T^{(k)}_{build} + \frac{\Delta_{GMRES}(i,j)}{\Delta_{GMRES}(1,1)} T^{(k)}_{GMRES}(1,1) \tag{14}$$

A first set of experiments has been executed with the aim of validating (14). These experiments are executed on clusters `IA-64-1` and `UniPart1`. Tables 1 and 2 report results concerning the denoising of a sequence of 14 frames of size 151×151×101. We show the Performance Contract $PC^{(0)}(1,j)$ related to node 0 for all values of j (the frames) and for $i = 1$ (one scale), and the execution time $T^{(0)}(1,j)$ (in seconds) for denoising the frame j. Further we show, in the last column, the relative error obtained estimating the actual execution time and that estimated by the Performance Contract. Note that the error is of 10% at most. Results refer only to node 0 because we did not observe significant differences with the other nodes of the clusters `IA-64-1` and `UniPart1`.

Table 1. Monitoring on the cluster IA-64-1

Frame index	$PC^{(0)}(1,j)$ (in secs.)	$T^{(0)}(1,j)$ (in secs.)	Relative error
1	20.9512	20.9512	0.00
2	14.4678	13.3022	0.08
3	14.7625	13.5491	0.08
4	15.6466	14.4584	0.08
5	16.5307	15.3111	0.07
6	16.2360	15.0253	0.07
7	15.6466	14.3392	0.08
8	14.7625	13.5789	0.08
9	14.4678	13.2639	0.08
10	15.6466	14.3577	0.08
11	15.3519	14.0668	0.08
12	15.6466	14.3602	0.08
13	15.6466	14.2467	0.09
14	19.7724	18.8719	0.05

Migration is a crucial task, because it depends on the relative overhead. Of course, such overhead may drastically change if migration occurs on nodes of the same resource or it is needed to migrate on another resource. In the Table 3 we report, in the first column the time (in seconds) needed to migrate on other nodes of the same cluster (we refer to the *local* migration), in the second column we report the time (in seconds) needed to migrate on different resources of the same geographic area, and we consider the two clusters located in Naples. Finally, in the third column, we report the time (in seconds) needed to migrate on the cluster located in Lecce (we refer to these last two cases as to a *remote* migration).

Table 2. Monitoring on the cluster UniPart1

Frame index	$PC^{(0)}(1,j)$ (in secs.)	$T^{(0)}(1,j)$ (in secs.)	Relative error
1	28.0687	28.0687	0.00
2	21.9950	20.3260	0.08
3	22.2711	20.6368	0.07
4	23.0993	20.9746	0.09
5	23.9275	21.5562	0.10
6	23.6515	21.2710	0.10
7	23.0993	20.8936	0.10
8	22.2711	20.1092	0.10
9	21.9950	19.9545	0.09
10	23.0993	20.8051	0.10
11	22.8232	20.7351	0.09
12	23.0993	20.7394	0.10
13	23.0993	20.8089	0.10
14	26.9644	25.4509	0.06

Table 3. Local vs. Remote Migration time (secs.)

Task	Local	Naples area	wide area
New resources selection	0	240	240
Data moving (1 frame=3MB)	0	6	0.2
Application starting	1	1	1

As expected, the overhead introduced by the resource brokering in case of remote migration, is much larger than that for the local migration. Therefore, let:

- T_{mo} be the migration overhead, as reported in Table 3;
- R_{old} be the execution time needed to terminate the algorithm on the initial resource;
- R_{new} be the execution time needed to terminate the algorithm on the resource where it migrates;

then, a migration on another resource occurs if

$$R_{new} + T_{mo} < R_{old}$$

where R_{old} is estimated by the Migration Manager as follows:

$$R_{old} = \widehat{T} \cdot nscales \cdot RF$$

where $\widehat{T}$ is the average time needed on for denoising the frames before the migration, and RF is the number of frames not yet denoised. To estimate R_{new} the Migration Manager evaluates the ratio between the benchmarks on the two systems, as defined in (13). More precisely, let B_{old} and B_{new} respectively denote the average benchmarks computed on all nodes of the initial resource and of the alternative resource, then, the Migration Manager computes:

$$B_{new/old} = \frac{B_{new}}{B_{old}}$$

and we assume that:

$$R_{new} \simeq B_{new/old} R_{old}$$

From Tables 1, 2 and 3, we observe that a remote migration should occur only if a strong violation of the Performance Contract occurs while the algorithm is processing the first two or three frames. Otherwise, once first frames have been processed, the overhead makes the remote migration not convenient. However, in our experiments, the overhead relative to the remote migration is mainly due to the execution of the benchmark needed for the selection of an alternative resource. This step is executed at runtime when the Monitor detects a contract violation. Of course, to reduce this overhead, the benchmark values should have been already available.

5 Future Works

We are currently working on the introduction of a fault-tolerance mechanism into the PETSc based application combined with some kind of checkpointing [12, 19]. We are using an *algorithm-based* approach relying on FT-MPI [15] which provides the software tools needed to identify and manage faults. We are using a disk based checkpointing method, indeed the algorithm already writes onto the disk the vectors u_j^i for each i and j. Moreover, to compute the vector u_j^i we only need u_{j-1}^{i-1}, u_j^{i-1} and u_{j+1}^{i-1}. Then, in order to recover from a fault we restart from those i and j corresponding to the last computed u_j^i, and the check of faults is performed at the end of each step of the innermost loop of the algorithm.

FT-MPI allows to re-spawn failed processes and to decide if to drop, or not, all ongoing messages. Moreover, when FT-MPI is used, the MPI context is redefined after each process is re-spawned. The main drawback seems to be the heavy dependence of all the PETSc global objects on the MPI context: i.e. the `PETSC_COMM_WORLD` macro, used by all PETSc objects, is a “copy” of the underlying `MPI_COMM_WORLD` MPI context. This suggests to address the fault-tolerance by the following steps:

- check the status of the processes: if a process has been re-spawned, then:
  - destroy the PETSc environment with all its objects,
  - re-inizialize the PETSc environment and create all the PETSc objects that are needed,
- restart the iterations from the last computed and saved u_j^i.

References

1. Arnold, D., S. Agrawal, S. Blackford, S., J. Dongarra, M. Miller, K. Seymour, K. and Sagi, K. and Shi, Z. and Vadhiyar, S. , *Users Guide to Netsolve* - Univ. of Tennessee Tech. Rep. ICL-UT-02-05, 2002.
2. Aydt R., C. Mendes, D. Reed, F. Vraalsen, *Specifying and Monitoring GRADS contracts*, http://hipersoft.cs.rice.edu/grads/publications/grid2001.pdf, 2001.
3. Balay S., K. Bushelman, W. Gropp, D. Kaushik, M. Knepley, L. Curfman McInnes, B. Smith, H. Zhang, *Petsc Users Manual*, ANL-95/11- Revision 2.1.3, Argonne National Laboratory, 2003.
4. Berman F., To Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnson, K. Kennedy, C. Kesselman, J. Mellor-Crummey, D. Reed, L. Torczon, R. Wolsky , *The Grads Project: Software support for High Performance Grid Applications* - Int. Journal on High Performance Applications. Vol 15 (2001), pp. 327-344.
5. Bertero M., P. Bonetto, L. Carracciuolo, L. D'Amore, A. Formiconi, M. R. Guarracino, G. Laccetti, A. Murli and G. Oliva , *A Grid-Based RPC System for Medical Imaging,* chapter of Parallel and Distributed Scientific and Engineering Computing: Practice and Experience, (Y. Pan and L T. Yang editors), Nova Science Publishers, 2003, pp. 177-190.
6. Boccia V., L. Carracciuolo, P. Caruso, L. D'Amore, G. Laccetti, A. Murli, *Sull'integrazione di un'applicazione basata su PETSc in ambiente di grid computing*, ICAR-NA-CNR Tech. Rep. TR-04-25, 2004.
7. Boccia V., L. D'Amore, M. Guarracino, G. Laccetti, *A Grid enabled PSE for Medical Imaging: Experiences on MedIGrid*, chapter of Computer Based Medical Systems CBMS 2005 , (A. Tsymbal and P. Cunningham editors), IEEE Press, 2005, pp. 529-536.
8. Bonetto P., G. Comis, A.R. Formiconi, M. Guarracino, *A new approach to brain imaging, based on an open and distributed environment*, Proceedings of 1st Int. IEEE EMBS Conference on Neural Engineering, 2003.
9. Carracciuolo L. , L. D'Amore, A. Murli, *Towards a parallel component for imaging in PETSc programming environment: A case study in 3-D echocardiography*, Parallel Computing 32, 2006, pp.67-83.
10. Caruso P., G. Laccetti, M. Lapegna, *A Performance Contract System in a Grid Enabling Component Based Programming Environment*, chapter of Advances in Grid Computing - EGC 2005 (P.M.A. Sloot et al., editors), Lecture Notes in Computer Science n. 3470, Springer, 2005, pp. 982-992.
11. Chan T.F., J. Shen and L. Vese , *Variational PDE Models in Image Processing*, Notices of American Mathematical Society, Vol. 50 n. 1, (2003) 14-26.
12. D'Amore L., F. Gregoretti, A. Murli, *Diskless algorithm-based checkpointing in a fault tolerant medical imaging application*, Conferenza SIMAI, 2004, and FIRB Grid.it Italian National Project, WP9 working note WP9-39, 2004.
13. Duff I.S., H.A. van der Vorst, *Preconditioning and Parallel Preconditioning*, in: J. Dongarra et al., Numerical Linear Algebra for High-Performance Computers (SIAM, Philadelphia, PA, 1998).
14. Elmroth, E., J. Tordsson, *A Grid Resource Broker Supporting Advance Reservation and Benchmark-Based Resource Selection*, chapter of Applied Parallel Computing. State of the Art in Scientific Computing (J. Dongarra, K. Madsen,

J. Wasniewski editors), Lecture Notes in Computer Science n. 3732, Springer, 2006, pp. 1061-1070.

15. Fagg G.E., A. Bukovsky , S. Vadhiyar, J. Dongarra, *Fault-tolerant MPI for the Harness metacomputing system*, Lecture Notes in Computer Science 2073:355–366.
16. Foster I. , C. Kesselman , *The Grid: Blueprint for a New Computing Infrastructure* - Morgan and Kaufman 1998
17. Guarracino M.R., G. Laccetti and A. Murli, *Application Oriented Brokering in a Medical Imaging: Algorithms and Software Architecture*, chapter of Advances in Grid Computing - EGC 2005 (P.M.A. Sloot et al., editors), Lecture Notes in Computer Science n. 3470, Springer, 2005, pp. 972-982.
18. Levesque R.J., *Finite Volume Methods for Hyperbolic Problems*, Cambridge University Press, New York, 2002.
19. Murli A. , L. D'Amore and F. Gregoretti , *I*/O Tolerance e Fault-Tolerance nell'algoritmo del gradiente coniugato, FIRB Grid.it Italian National Project, WP9 working note WP9-28, 2004.
20. Reed D.A. , R. Ribler, H. Simitci, J. S. Vetter, *Autopilot: Adaptive Control of Distributed Applications*, Proceedings of the Seventh IEEE International Symposium on High Performance Distributed Computing (HPDC), 1998.
21. Sapiro G., *Geometric Partial Differential Equations and Image Processing*, Cambridge University Press, New York, 2001.
22. Sarti A. , K. Mikula, F. Sgallari, *Nonlinear Multiscale Analysis of Three-Dimensional Echocardiographic Sequences*, IEEE Transactions on Medical Imaging, Vol. 18, N. 6 (1999), pp. 453-466.