A Scalable Numerical Algorithm for Solving Tikhonov Regularization Problems

Rosella Arcucci[1,2,4], Luisa D'Amore[1,2](✉), Simone Celestino[1], Giuliano Laccetti[1], and Almerico Murli[2,3]

[1] University of Naples Federico II, Naples, Italy
luisa.damore@unina.it
[2] Euro Mediterranean Centre on Climate Change (CMCC), Lecce, Italy
[3] SPACI, Naples, Italy
[4] Imperial College London, London, UK

Abstract. We present a numerical algorithm for solving large scale Tikhonov Regularization problems. The approach we consider introduces a splitting of the regularization functional which uses a domain decomposition, a partitioning of the solution and modified regularization functionals on each sub domain. We perform a feasibility analysis in terms of the algorithm and software scalability, to this end we use the scale-up factor which measures the performance gain in terms of time complexity reduction. We verify the reliability of the approach on a consistent test case (the Data Assimilation problem for oceanographic models).

Keywords: Tikhonov regularization · Large scale inverse problems · Parallel algorithm · Data assimilation

1 Introduction and Motivation

The solution of large scale inverse and ill posed problems arises in a variety of applications, such as those in the earth/climate science, including earth observation (remote sensing) and data assimilation [8,14], or those arising in image analysis, including medical imaging, astronomical imaging and restoration of digital films [2,4,5,9,10,15]. A straightforward solution of such problems is meaningless because the computed solution would be dominated by errors. Therefore some regularization must be employed. In this paper we focus on the standard Tikhonov Regularization (TR) method [16]. The efficient solution of TR problems critically depends on suitable numerical algorithms. Several strategies have been proposed in the literature. Basically, the approaches are based on the Conjugate Gradient iterative method, or on the Singular Value Decomposition. However, because of their formulation, these approaches are intrinsically sequential and none of them is able to address in an acceptable computational time large scale applications. For such simulations we need to address methods which allow us to reduce the problem to a finite sequence of sub problems of a more manageable size, perhaps without sacrificing the accuracy of the computed solution. Indeed, we need to employ scalable parallel algorithms.

R. Wyrzykowski et al. (Eds.): PPAM 2015, Part II, LNCS 9574, pp. 45–54, 2016.
DOI: 10.1007/978-3-319-32152-3_5

Here, scalability refers to the capability of the algorithm to:

– exploit performance of emerging computing architectures in order to get a solution in a suitable acceptable time (strong scaling),
– use additional computational resources effectively to solve increasingly larger problems (weak scaling).

In the present work we introduce a computational model which starts from a decomposition of the global domain into sub domains. On these sub domains we define local regularization functionals such that the minimum of the global regularization functional can be obtained by collecting the minimum of each local functional. The (global) problem is decomposed into (local) sub problems in such a way. The resulted algorithm consists of several copies of the original one, each one requiring approximately the same amount of computations on each sub domain and an exchange of boundary conditions between adjacent sub domains. The data is flowing across the surfaces, the so called surface-to-volume effect is produced.

A research collaboration between us, the Argonne National Laboratory in Chicago, the Imperial College London, the University of California Santa Cruz, and the Barcelona Supercomputing Center, within the H2020-MSCA-RISE-2015 Project NASDAC (iNnovative Approaches for Scalable Data Assimilation in oCeanography) give us the opportunity to work on variational Data Assimilation (DA) in Oceanographic Models [7,9]. Then we applied this approach to the (DA) inverse problem which is ill posed and variational approaches used for solving it are essentially derived from the TR formulation.

2 Preliminary Concepts

Here we introduce some notations we use in the next sections. For more details see [6].

Definition 1 (The Inverse problem). *Given the linear operators* $\mathbf{M} \in \Re^{N\times N}$ *and* $\mathbf{H} \in \Re^{S\times N}$*, and the vector* $\mathbf{v} \in \Re^{S\times 1}$*, where* $N >> S$*. Assume that* $\mathbf{H}$ *is highly ill conditioned. To compute* $\mathbf{u} : \Omega \mapsto \Re^{N\times 1}$ *such that*

$$\mathbf{v} = \mathbf{H}[\mathbf{u}] \tag{1}$$

subject to the constraint $\mathbf{u} = \mathbf{u}^{\mathbf{M}}$ *where* $\mathbf{u}^{\mathbf{M}} = \mathbf{M}[\mathbf{u}]$*.* ♠

The TR approach provides the approximation $\mathbf{u}(\lambda)$ of $\mathbf{u}$, where λ is the regularization parameter, as follows [13]

Definition 2 (The TR problem). *To compute*

$$\mathbf{u}(\lambda) = argmin_{\mathbf{u}} J(\mathbf{u}) \tag{2}$$

where

$$J(\mathbf{u}) = \|\mathbf{H}\mathbf{u} - \mathbf{v}\|_{\mathbf{R}}^2 + \lambda\|\mathbf{u} - \mathbf{u}^{\mathbf{M}}\|_{\mathbf{B}}^2, \tag{3}$$

is the TR problem of (1) and, where $\|\cdot\|_{\mathbf{B}}$ *and* $\|\cdot\|_{\mathbf{R}}$ *denote the weighted norms with respect to the error covariance matrices* $\mathbf{B}$ *and* $\mathbf{R}$ *and* λ *is the regularization parameter.* ♠

Definition 3 (Domain Decomposition). *Let*

$$\Omega = \bigcup_{i=1}^{p} \Omega_i \quad \Omega_i \subset \Re^3 \tag{4}$$

be the decomposition of $\Omega \subset \Re^3$ *where* $\Omega_i \subset \Re^3$ *are such that* $\Omega_i \cap \Omega_j = \Omega_{ij} \neq \emptyset$ *when the subdomains are adjacent.* ♠

Starting from a decomposition of the domain Ω, we now introduce the *local* TR functionals. A local TR functional, which describes the local problems on each sub-domain Ω_i, is obtained from the TR functional J in (3), by adding a *local* constraint defined on the overlapping regions in Ω_{ij}. This is in order to enforce the continuity of each solution of the local DA problem onto the overlap region between adjacent domains Ω_i and Ω_j.

Definition 4 (Local TR functional). *Let* $\mathbf{H}_i, \mathbf{u}^i, \mathbf{v}^j, (\mathbf{u}^{\mathbf{M}})^i, \mathbf{R}_i$ *and* $\mathbf{B}_i$*, be the restrictions on* Ω_i *of* $\mathbf{H}, \mathbf{u}, \mathbf{v}$ *and* $\mathbf{u}^M, \mathbf{R}$ *and of* $\mathbf{B}$*, respectively. Let* $\mathbf{u}^j$ *be the restriction on* Ω_j *of* $\mathbf{u}$*,* $\mathbf{B}_{ij}$ *be the restriction of* $\mathbf{B}$ *on the overlapping region* Ω_{ij}*. Finally, let* λ_i *and* ω_i *be the (local) regularization parameters. Then*

$$\mathbf{u}^i(\lambda_i, \omega_i) = argmin_{\mathbf{u}^i} J(\Omega_i, \lambda_i, \omega_i)$$

where

$$\begin{aligned} J(\Omega_i, \lambda_i, \omega_i) = \|\mathbf{H}_i \mathbf{u}^i - \mathbf{v}^i\|^2_{\mathbf{R}_i} + \lambda_i \|\mathbf{u}^i - (\mathbf{u}^{\mathbf{M}})^i\|^2_{\mathbf{B}_i} \\ + \omega_i \|\mathbf{u}^i/\Omega_{ij} - \mathbf{u}^j/\Omega_{ij}\|^2_{\mathbf{B}_{ij}} \end{aligned} \tag{5}$$

is the minimum of the local TR functional $J(\Omega_i, \lambda_i, \omega_i)$*.* ♠

In [6] the authors proved that

$$\mathbf{u}(\lambda) = \sum_{i=1,p} \mathbf{u}^{EO_i}(\lambda_i, \omega_i), \tag{6}$$

where

$$\mathbf{u}^{EO_i}(\lambda_i, \omega_i) : \Omega_i \mapsto \Omega$$

and

$$\mathbf{u}^{EO_i}(\lambda_i, \omega_i) := \begin{cases} \mathbf{u}^i & \textit{on } \Omega_i \\ 0 & \textit{elsewhere} \end{cases}$$

This result states that the minimum of J, in (2), can be regarded as a piecewise function obtained by patching together $\mathbf{u}^i$, i.e. the minimum of the operators $J(\Omega_i, \lambda_i, \omega_i)$; it means that, by using the domain decomposition, the global minimum of the operator J can be obtained by patching together the minimum of the *local* functionals $J(\Omega_i, \lambda_i, \omega_i)$.

In the following we refer to the decomposition of TR functional as the DD-TR model.

2.1 The Algorithmic Scalability

Large-scale problems are computationally expensive and their solution requires designing of scalable approaches. Many factors contribute to scalability, including the architecture of the parallel computer and the parallel implementation of the algorithm. However, one important issue is the scalability of the algorithm itself. We use the following measure

Definition 5 (Scalable Algorithm). *If $p \in \mathcal{N}$, and $p > 1$, the algorithm associated to the decomposition given in (4) is*

$$\mathcal{A}(\Omega, p) := \{\mathcal{A}(\Omega_1), \mathcal{A}(\Omega_2), \ldots, \mathcal{A}(\Omega_p)\}$$

where $\mathcal{A}(\Omega_i)$ is the local algorithm on Ω_i. ♠

Definition 6 (Scale up factor). *Let $p_1, p_2 \in \mathcal{N}$ and $p_1 < p_2$. Let $T(\mathcal{A}(\Omega, p_i))$, $i = 1, 2$ denote the time complexity of $\mathcal{A}(\Omega_i, p_i)$, $i = 1, 2$. $\forall\ i \neq j$ we define the (relative) scale up factor of $\mathcal{A}(\Omega, p_2)$, in going from p_1 to p_2, the following ratio:*

$$S_{p_2,p_1}(N) = \frac{T(\mathcal{A}(\Omega, p_1))}{(p_2/p_1)T(\mathcal{A}(\Omega, p_2))}.$$

♠

We observe that:

1. if N is fixed and $p \sim N$ we get the so called *strong scaling*.
2. if $N \to \infty$ and r is kept fixed, then we get the so called *weak scaling*.

3 The Case Study

Let $t \in [0, T]$ denote the time variable. Let $u^{true}(t, x)$ be the evolution state of a predictive system governed by the mathematical model $\mathcal{M}$ with $u^{true}(t_0, x)$, $t_0 = 0$ as initial condition. Here we consider a 3D shallow water model. Let $v(t, x) = \mathcal{H}(u^{true}(t, x))$ denote the observations mapping, where $\mathcal{H}$ is a given nonlinear operator which includes transformations and grid interpolations. According to the real applications of model-based assimilation of observations, we will use the following definition of Data Assimilation (DA) inverse problem [13,14]. Given

- $D_N(\Omega) = \{x_j\}_{j=1,\ldots,N} \in \Re^N$: a discretization of $\Omega \subset \Re^3$;
- $\mathbf{M}$: a discretization of $\mathcal{M}$;
- $\mathbf{u}_0^{\mathcal{M}} = \{u_0^j\}_{j=1,\ldots,N}^{\mathcal{M}} \equiv \{u(t_0, x_j)\}_{j=1,\ldots,N}^{\mathcal{M}} \in \Re^N$: numerical solution of $\mathcal{M}$ on $D_N(\Omega)$. This is the background estimates, i.e. the initial states at time t_0; it is assumed to be known, usually provided by a previous forecast.
- $\mathbf{u}^{\mathbf{M}} = \{u^j\}_{j=1,\ldots,N} \equiv \{u(x_j)\}_{j=1,\ldots,N} \in \Re^N$: numerical solution of $\mathbf{M}$ on $D_N(\Omega)$;
- $\mathbf{u}^{true} = \{u(x_j)^{true}\}_{j=1,\ldots,N}$: the vector values of the reference solution of $\mathcal{M}$ computed on $D_N(\Omega)$ at t fixed;

- $\mathbf{v} = \{v(y_j)\}_{j=1,\ldots,nobs}$: the vector values of the observations on $D_N(\Omega)$;
- $\mathcal{H}(x) \simeq \mathcal{H}(z) + \mathbf{H}(x-z)$: where $\mathbf{H} \in \Re^{N \times nobs}$ is the matrix obtained by the first order approximation of the Jacobian of $\mathcal{H}$ and $nobs << N$;
- $\mathbf{R}$ and $\mathbf{B}$ the covariance matrices of the errors on the observations $\mathbf{v}$ and on the system state $\mathbf{u^M}$, respectively. These matrices are symmetric and positive definite (see [6] for details).

We assume that $N = n_x \times x_y \times n_z$ and $n_x = n_y = n$ while $n_z = 3$. Since the unknown vectors are the fluid height or depth, and the two-dimensional fluid velocity fields, the problem size is $N = 3n^2$. $\mathbf{H}$ is assumed to be a piecewise linear interpolation operator whose coefficients are computed using the points of model domain nearest the observation values. We assume $\mathbf{u}^{true}$ be the solution of $\mathcal{M}$ as given in [1]. Observation values are randomly chosen among the values of $\mathbf{u}^{true}$.

Definition 7 (The DA Inverse problem). *Let* $\mathbf{u}^{DA}$ *be the solution of:*

$$\mathbf{v} = \mathbf{H}[\mathbf{u}^{DA}]$$

subject to the constraint:

$$\mathbf{u}^{DA} = \mathbf{u^M}. \qquad \spadesuit$$

DA is an ill posed inverse problem [14]. The local DD-TR operator, defined on a subdomain Ω_i, is (see Eq. (5), with $\lambda_i = \omega_i = 1$)):

$$\begin{aligned} J_i(\mathbf{u}^i) &= (\mathbf{H}_i\mathbf{u}^i - \mathbf{v}^i)^T\mathbf{R}_i(\mathbf{H}_i\mathbf{u}^i - \mathbf{v}^i) + (\mathbf{u}^i - (\mathbf{u^M})^i)^T\mathbf{B}_i(\mathbf{u}^i - (\mathbf{u^M})^i) \\ &\quad + (\mathbf{u}^i - \mathbf{u}^j)^T\mathbf{B}_{ij}(\mathbf{u}^i - \mathbf{u}^i). \end{aligned} \tag{7}$$

In [3,7] the authors provided the reliability of DD-TR model for DA problem. In this paper we present results of an implementation of the model on two different computing architectures. We evaluate the efficiency of these implementations by analysing the strong and weak scalability of the algorithm by using the scale up factor defined in Sect. 2.1.

4 The DD-TR Algorithm on Two Reference Computing Architectures

In this paper, our testbed is a distributed computing environment composed of computational resources, located in the University of Naples Federico II campus, connected by local-area network. More precisely, the testbed is made of:

- A1: a 288 CPU-multicore architecture made of distributed memory blades each one with computing elements sharing the same local memory for a total of 3456 cores.
- A2: a GPU+CPU architecture made of the 512 threads NVIDIA Tesla connected to a quad-core CPU.

If $nproc$ denotes the number of processing elements of the reference architectures, we have $nproc = 64$ for A1, and $nproc = \#$ threads-blocks, for A2. We assume a 2D uniform decomposition of $D_N(\Omega)$ along the (x, y)-axis, that is the x-axis is divided by s and the y-axis by q then, the size of each subdomain $D_N(\Omega_i)$ is $r = nloc_x \times nloc_y \times nloc_z$ where:

$$nloc_x = \frac{n_x}{s} + 2o_x \ , \ nloc_y = \frac{n_y}{q} + 2o_y \ \ , \ nloc_z = n_z. \tag{8}$$

These dimensions include the overlapping $(2o_x \times 2o_y)$.

We use the LBFGS method for computing the minimum of DD-TR functionals [11,17]. Then, following result specifies the scale up factor of algorithm $\mathcal{A}(D_N(\Omega), p)$ in our case study [6]:

Proposition 1. *If the time complexity of* $\mathcal{A}(D_N(\Omega), 1)$ *is* $T(N) = O(f(N))$ *flops, on a problem of size* N*, where* $f(N) \in \Pi_3$*, the scale up factor of the algorithm* $\mathcal{A}(D_N(\Omega), p)$ *is*

$$S_{p,1}(N) = \alpha(r, p)\, p^2. \tag{9}$$

Remark: Let t_{flop} denote the unitary time required by one floating point operation. As a result, the execution time needed to algorithm $\mathcal{A}(D_N(N), 1)$ for performing $T(N)$ floating point operations, is

$$T_{flop}(N) = T(N) \times t_{flop}.$$

Multiplying and dividing the (9) by t_{flop} we get

$$\alpha(r, p)p^2 = \frac{T_{flop}(N)}{pT_{flop}(N/p)}. \tag{10}$$

Finally, we give the following

Definition 8. *Let* $\frac{S}{V} := \frac{T_{oh}(N/p)}{T_{flop}(N/p)}$ *denote the surface-to-volume ratio. It is a measure of the amount of data exchange (proportional to surface area of domain) per unit operation (proportional to volume of domain).* ♠

In [12] authors define $T^{nproc}(N)$, the execution time of $\mathcal{A}(N, p)$, as given by time for computation plus an overhead which is given by synchronization, memory accesses and communication time also.

$$T^{nproc}(N) := T^{nproc}_{flop}(N) + T^{nproc}_{oh}(N)$$

where

- A1: $T_{flop}^{nproc}(N)$ is computing time required for the execution of $T(N)$ floating point operations; $T_{oh}^{nproc}(N)$ is overhead time of $T(N)$ data which includes communications among CPU processors.
- A2: $T_{flop}^{nproc}(N) := T^{CPU}(N) + T^{GPU}(N)$, where
  - $T^{CPU}(N)$ is the CPU execution time for the execution of $T(N)$ floating point operations,
  - $T^{GPU}(N)$ is the GPU execution time for the execution of $T(N)$ floating point operations.

  and $T_{oh}^{nproc}(N)$ includes the communications time between host (CPU) and device (GPU) and time for memories accesses.

  Here we assume that

$$T^{GPU}(N) := T_{flop}^{GPU}(N) + T_{mem}^{GPU}(N), \tag{11}$$

  where $T_{mem}^{GPU}(N)$ is the time for global and local memories transfers into the device (GPU) and $T_{flop}^{GPU}(N)$ is the computing time required for execution of floating point operations.

Finally, for A2, $T_{oh}^{nproc}(N) \equiv T_{mem}^{GPU}(N)$, since the communications between host (CPU) and device (GPU) in the algorithm we implement occur just at the begin and the end for I/O transfers and, for this reason, it can be neglected in our considerations.

4.1 Discussion

Table 1 shows results obtained for $\mathcal{A}(\Omega, p)$ on A1 for a problem size $O(10^6)$ and $O(10^7)$ by using $nproc = p$ and Table 2 shows execution time of the algorithm $\mathcal{A}(\Omega, p)$ running on A2 for a problem size $O(10^7)$ by using # thread-blocks= $2p$.

In Table 2, $T^{CPU}(N)$ is execution time that CPU needs for building data. These data are transferred just once as well as output data so we have that $T_{oh}^{GPU}(N)$ is reduced to the time of I/O transfer. For this reasons we evaluate the performance of DD-TR implementation on GPU by analysing $T^{GPU}(N)$. $T_{oh}(N)$ can be estimates by dividing D_N, which is size of processed data espressed in GB by the bandwidth value B_W which is the rate of data transfer espressed in GB/seconds: $T_{oh}(N) := \frac{D_N}{B_W}\ secs$.

We have $D_N = 3.7\,\text{GB}$ which gives $T_{oh} \simeq 3.7/208\,\text{s} \simeq 0.017\,\text{s}$. Our considerations will focus on values of $T_{flop}^{GPU}(N)$ reported in Table 3. We now discuss the software scalability as shown in Tables 1 and 3. To this end, we introduce

$$s_{nproc}^{loc} := \frac{T_{flop}(N/p)}{T_{nproc}(N/p)}, \tag{12}$$

which denotes the speed up of the (local) algorithm $\mathcal{A}(D_N(\Omega_i), N/p)$ for solving the local problem on subdomain $D_N(\Omega_i)$. Let us express the measured scale up

Table 1. Results on A1: Execution time and *scale up* factor of $\mathcal{A}(\Omega, p)$ for different values of $N = 3n^2$ and $nproc = 2p$.

n	$nproc$	$T^{nproc}(N)$	Measured $S_{nproc,8}$	$S_{nproc,8}$
$O(10^6)$	8	2.0545e+02	1.0	1
	16	6.3316e+01	3.25	4
	32	2.0005e+01	10.27	16
	64	8.7835e+00	23.39	64
n	$nproc$	$T^{nproc}(N)$	Measured $S_{nproc,16}$	$S_{nproc,16}$
$O(10^7)$	8	–	–	–
	16	3.9091e+03	1.0	1
	32	9.9952e+02	3.91	4
	64	2.7584e+02	14.17	16

Table 2. Execution time of algorithm $\mathcal{A}(\Omega, p)$ running on A2 for a problem size $O(10^7)$ and $nproc = \#thread - blocks$.

N	p	$nproc$	$T^{GPU}(N)$
$O(10^7)$	1	2	0.144
	2	4	0.044
	4	8	0.025
	8	16	0.024

factor in terms of s^{loc}_{nproc}. We have:

$$S^{measured}_{1,nproc} := \frac{T_{flop}(N)}{p \cdot (T_{flop}(r_i) + T_{oh}(N/p))}. \tag{13}$$

From the (12) and the (13) it follows that

$$S^{measured}_{1,nproc} = \frac{T_{flop}(N)}{\frac{pT_{flop}(N/p)}{s^{loc}_{nproc}} + pT_{oh}(N/p)} = \frac{s^{loc}_{nproc} \frac{T_{flop}(N)}{pT_{flop}(N/p)}}{1 + \frac{s^{loc}_{nproc} T_{oh}(N/p)}{T_{flop}(N/p)}}. \tag{14}$$

Table 3. Results on A2: Values of T^{GPU}_{flop} and measured *scale up*factor compared with theoretical once.

N	p	$T^{GPU}_{flop}(N)$	Measured $S_{nproc,2}$	$S_{nproc,2}$
$O(10^7)$	1	0.127	-	-
	2	0.027	4.7	4
	4	0.008	15.9	8
	8	0.007	18.1	16

As we need to guarantee that the so-called *surface-to-volume* effect on each local DA problem is produced [2,4,9,10], we assume:

$$0 \leq \frac{S}{V} < 1 - \frac{1}{s^{loc}_{nproc}} < 1.$$

Let

$$\alpha := \frac{s^{loc}_{nproc}}{1 + \frac{s^{loc}_{nproc} T_{oh}(N/p)}{T_{flop}(N/p)}} = \frac{s^{loc}_{nproc}}{1 + s^{loc}_{nproc}\frac{S}{V}},$$

from (14) it comes out that

$$S^{measured}_{1,nproc} = \alpha S_{1,nproc}.$$

Finally, it holds that

(i) if $s^{loc}_{nproc} = 1$ then

$$\alpha < 1 \Leftrightarrow S^{measured}_{nproc,1} < S_{nproc,1};$$

(ii) if $s^{loc}_{nproc} > 1$ then

$$\alpha > 1 \Leftrightarrow S^{measured}_{nproc,1} > S_{nproc,1};$$

(iii) if $s^{loc}_{nproc} = p$ then

$$1 < \alpha < p \Rightarrow S^{measured}_{nproc,1} < pS_{nproc,1};$$

Hence, we may conclude that if

$$s^{loc}_{nproc} \in]1, p] \Rightarrow S^{measured}_{nproc} \in]S_{nproc,1}, p\, S_{nproc,1}[.$$

It is worth noting that in our experiments, in A1, local DA problems are sequentially solved, then

$$s^{loc}_{nproc} = 1$$

while in A2, local DA problems have been concurrently solved on the GPU device, so

$$s^{loc}_{nproc} > 1$$

Thus the above analysis validates the experimental results both in terms of strong and weak scaling.

Acknowledgments. This work has been realised thanks to the use of the SCoPE computing infrastructure at the University of Naples, also in the framework of PON *"Rete di Calcolo per SuperB e le altre applicazioni"* (ReCaS) project. This work was developed within the research activity of the H2020-MSCA-RISE-2016 NASDAC Project N. 691184.

References

1. Moler, C.: Experiments with MATLAB (2011)
2. Antonelli, L., Carracciuolo, L., Ceccarelli, M., D'Amore, L., Murli, A.: Total variation regularization for edge preserving 3D SPECT imaging in high performance computing environments. In: Sloot, P.M.A., Tan, C.J.K., Dongarra, J., Hoekstra, A.G. (eds.) ICCS-ComputSci 2002, Part II. LNCS, vol. 2330, p. 171. Springer, Heidelberg (2002)
3. Arcucci, R., D'Amore, L., Carracciuolo, L.: On the problem-decomposition of scalable 4D-Var data assimilation models. In: International Conference on High Performance Computing & Simulation (HPCS 2015), pp. 589–594 (2015)
4. Carracciuolo, L., D'Amore, L., Murli, A.: Towards a parallel component for imaging in PETSc programming environment: a case study in 3-D echocardiography. Parallel Comp. **32**(1), 67–83 (2006)
5. D'Amore, L., Murli, A.: Regularization of a Fourier series method for the Laplace transform inversion with real data. Inverse Prob. **18**(4), 1185–1205 (2002)
6. D'Amore, L., Arcucci, R., Carracciuolo, L., Murli, A.: A scalable approach to variational data assimilation. J. Sci. Comput. **61**, 239–257 (2014)
7. D'Amore, L., Arcucci, R., Carracciuolo, L., Murli, A.: DD-OceanVar: a domain decomposition fully parallel data assimilation software in mediterranean sea -. Procedia Computer Science **18**, 1235–1244 (2013)
8. D'Amore, L., Arcucci, R., Marcellino, L., Murli, A.: A parallel three-dimensional variational data assimilation scheme. AIP Conf. Proc. **1389**(1), 1829–1831 (2011)
9. D'Amore, L., Arcucci, R., Marcellino, L., Murli, A.: HPC computation issues of the incremental 3D variational data assimilation scheme in OceanVar software. J. Numer. Anal. Ind. Appl. Math. **7**(3–4), 91–105 (2012)
10. D'Amore, L., Casaburi, D., Galletti, A., Marcellino, L., Murli, A.: Integration of emerging computer technologies for an efficient image sequences analysis. Integr. Comput. Aided Eng. **18**(4), 365–378 (2011)
11. D'Amore, L., Laccetti, G., Romano, D., Scotti, G., Murli, A.: Towards a parallel component in a GPU–CUDA environment: a case study with the L-BFGS Harwell routine. Int. J. Comput. Math. **92**(1), 59–76 (2015)
12. Flatt, H.P., Kennedy, K.: Performance of parallel processors. Parallel Comput. **12**, 1–20 (1989)
13. Haben, S.A., Lawless, A.S., Nichols, N.K.: Conditioning of the 3DVAR Data Assimilation Problem, Mathematics Report 3/2009. University of Reading, Department of Mathematics (2009)
14. Kalnay, E.: Atmospheric Modeling, Data Assimilation and Predictability. Cambridge University Press, Cambridge (2003)
15. Murli, A., Cuomo, S., D'Amore, L., Galletti, A.: Numerical regularization of a real inversion formula based on the Laplace transform's eigenfunction expansion of the inverse function. Inverse Prob. **23**(2), 713–731 (2007)
16. Tikhonov, A.N., Arsenin, V.Y.: Solutions of Ill-Posed Problems. Wiley, New York (1977)
17. Zhu, C., Byrd, R.H., Lu, P., Nocedal, J.: Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. ACM TOMS **23**(4), 550–560 (1997)