Int J Parallel Prog (2017) 45:1214–1235
DOI 10.1007/s10766-016-0460-3

A Decomposition of the Tikhonov Regularization Functional Oriented to Exploit Hybrid Multilevel Parallelism

Rossella Arcucci[1,2] · Luisa D'Amore[1,2] · Luisa Carracciuolo[3] · Giuseppe Scotti[1] · Giuliano Laccetti[1]

Received: 1 March 2016 / Accepted: 22 September 2016 / Published online: 11 October 2016

Abstract We introduce a decomposition of the Tikhonov Regularization (TR) functional which split this operator into several TR functionals, suitably modified in order to enforce the matching of their solutions. As a consequence, instead of solving one problem we can solve several problems reproducing the initial one at smaller dimensions. Such approach leads to a reduction of the time complexity of the resulting algorithm. Since the subproblems are solved in parallel, this decomposition also leads to a reduction of the overall execution time. Main outcome of the decomposition is that the parallel algorithm is oriented to exploit the highest performance of parallel architectures where concurrency is implemented both at the coarsest and finest levels of granularity. Performance analysis is discussed in terms of the algorithm and software scalability. Validation is performed on a reference parallel architecture made of a distributed memory multiprocessor and a Graphic Processing Unit. Results are presented on the Data Assimilation problem, for oceanographic models.

✉ Luisa D'Amore
luisa.damore@unina.it

Rossella Arcucci
rossella.arcucci@unina.it

Luisa Carracciuolo
luisa.carracciuolo@cnr.it

Giuseppe Scotti
giuseppe.scotti@unina.it

Giuliano Laccetti
giuliano.laccetti@unina.it

[1] University of Naples Federico II, Naples, Italy

[2] Euro Mediterranean Center on Climate Changes (CMCC), Lecce, Italy

[3] Istituto per i Polimeri, Compositi e Biomateriali of the CNR (IPCB-CNR), Naples, Italy

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

1 Introduction and Motivation

The solution of large scale linear discrete ill posed problems arise in a variety of applications, such as those in the earth/climate science, including earth observation (remote sensing) and data assimilation [2,7], or those arising in image analysis, including medical imaging, astronomical imaging and restoration of digital films [1,4,9,10,12–14,26] and those arising in solving the Laplace transform integral equation [3]. Upon discretization, linear systems such as the following

$$\mathbf{Tw} = \mathbf{b}, \quad \mathbf{T} \in \Re^{N\times M}, \quad \mathbf{w} \in \Re^{M\times 1}, \quad \mathbf{b} \in \Re^{N\times 1}, \tag{1}$$

where $M \geq N$, must be solved. In this setting the coefficient matrix $\mathbf{T}$ has typically an ill determined numerical rank, i.e. the singular values decay to zero with no evident gap between two consecutive ones to indicate the numerical rank; in particular, $\mathbf{T}$ is ill conditioned. Moreover the available right hand side vector $\mathbf{b}$ is affected by some measurement errors (which we will refer to as noise), i.e.

$$\mathbf{b} = \mathbf{b}^{ef} + \mathbf{e}, \tag{2}$$

where $\mathbf{b}^{ef} \in \Re^{N\times 1}$ represents the unknown error free right hand side and $\mathbf{e} \in \Re^{N\times 1}$ the additive noise vector.

A straightforward solution of (1) 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. This approach improves the condition of (1) by solving the constrained least square problem

$$\|\mathbf{Tw} - \mathbf{b}\|_2 + \lambda\|\mathbf{Qw}\|_2 = \min \tag{3}$$

where $\mathbf{Q}$ is referred to as the regularization matrix and the scalar λ is known as the regularization parameter. The matrix $\mathbf{Q}$ is commonly chosen to be the identity matrix; however, if the desired solution has particular known properties, then it may be meaningful to let $\mathbf{Q}$ be a scaled finite difference approximation of a differential operator or a scaled orthogonal projection. Finally, $\|\cdot\|_2$ denotes the L^2-norms in $\Re^N$. For an introduction to the solution of this kind of problems we refer to [36].

The solution of (3) critically depends on suitable numerical algorithms. Several strategies have been proposed in the literature. Basically, the approaches are based on the Conjugate Gradient (CG) iterative method [19,35], or on the Singular Value Decomposition(SVD) [20]. 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 TR 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. Here, scalability refers to the capability of the algorithm to:

- Exploit performance of emerging computing architectures in order to minimise the time to solution for a given problem with a fixed dimension (strong scaling),
- Use additional computational resources effectively to solve increasingly larger problems (weak scaling).

1.1 Contribution of the Work

As it is highlighted in the ETP4HPC Strategic Research Agenda recommendations [16], the key architectural changes of exascale machines will force changes throughout the overall layers of the so called *mathematics stack*—including the mathematical modeling—in ways that cannot be completely hidden from the associated numerical solvers situated at the bottom layer of the math stack.
In line with such computational needs, here we propose a parallel algorithm that is designed to exploit the multilevel parallelism of emerging architectures. More precisely, we introduce a computational method which starts from a decomposition of the global domain into overlapped smaller sub domains. On these sub domains we define local TR problems and we prove that the solution of the (global) TR problem can be obtained by collecting the solutions of each local problem. 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. The equations in the subdomains are solved and the matching of the solutions is imposed iteratively. Since the computational cost of the numerical algorithm is nonlinear w.r.t. the number of unknowns, breaking the initial problem into a set of smaller subproblems is profitable.

Innovation goes from the introduction of multiple levels of parallelism depending on the granularity of the operations. In other words, it arises the structure of a "granular polyalgorithm" in which, following a tree configuration manner, its parallelism takes into account the degree of dependence of operations [22,25]. Therefore, instead of increasing the number of subdomains to be assigned to each computing element, the domain decomposition follows a tree configuration grafting a level of decomposition in the previous. This ensures to keep small the number of subdomains generated at the first level (coping with the scalability of local numerical solvers which depends on the surface-to-volume effect) and at the same time, distributing further subdomains within each processing element, we introduce levels of parallelism in a hierarchical manner (meeting the scalability of parallel software, which depends on the exploitation of the performance of the emerging architectures). We validate this approach in a Data Assimilation (DA) problem, which is an ill posed inverse problem [8,9,27].
As we shall discuss in detail in this paper the performance gain that we get from this approach is two fold:

- Instead of solving one larger problem we can solve several smaller problems which leads to a reduction of each *local* algorithm's time complexity,

– The subproblems reproduce the problem at smaller dimensions and they are solved in parallel, which leads to a reduction of the *global* software execution time.

To analyse the reduction of the algorithm's time complexity, we will use the scale-up factor while for quantifying the reduction of the software execution time we will use the measured scale-up [6].

1.2 Related Works

For solving linear algebra problems, since the early 90' parallel algorithms have been developed, mainly introducing block or partitioned[1] formulations of the basic linear algebra operations and suitable data distribution schemes. MAGMA, PLASMA, pARMS are well known software libraries which result from this investment [24,31,34]. Moreover, the PETSc software library [33] has a parallel solver for the LSQR [30], which is one of the most effective algorithms for solving least square problems. For efficiently solving large scale TR problems, where $\mathbf{T}$ is large and sparse, numerical approaches that combine the LSQR algorithm with regularization are developed in [5,29]. These formulations appear to be more attractive for an effective implementation on advanced architectures, increasing the granularity of the underlying scalar algorithm.
Nevertheless, one common drawback of the aforementioned parallel approaches is that parallel approaches which exploit only a single type of parallelism have clear performance limitations (such as synchronization points, sequential parts, etc.), that prevent effective scaling with the thousands of processors available in massively parallel computers. This motivated the development of multilevel or multigrained parallel algorithms. The present work attempts to contribute to the design and development of such a scalable parallel algorithm.

1.3 Organization of the Article

The article is organized as follows. In Sect. 2 we introduce main concepts and definitions related to the domain decomposition method for the solution of Tikhonov regularization problems. In Sect. 3 we illustrate how to split the problem into lower dimensional problems and we prove that the minimum of the global regularization functional can be obtained by collecting the minimum of each local functional. Section 3.1 provides a case study on the Data Assimilation problem. Section 4 reports the scalability metrics which will be used to discuss the performance results. In Sect. 4 we describe and analyse the algorithm we have developed and finally, conclusions are given in Sect. 6.

[1] A partitioned algorithm is a scalar (or point) algorithm in which the operations have been grouped and reordered into matrix operations. A block algorithm is a generalization of a scalar algorithm in which the basic scalar operations become matrix operations, and a matrix property based on the nonzero structure becomes the corresponding property blockwise (LAPACK contains only partitioned algorithms that is, the main computations are block oriented and implemented by using BLAS-3) [15].

2 Preliminary Concepts

In this section we introduce some mathematical concepts and notations we need to use in the next sections.

Definition 1 (*Domain decomposition*) Let Ω be a finite numerable set such that $card(\Omega) = N$. Let

$$\Omega = \bigcup_{i=1}^{p} \Omega_i, \quad card(\Omega_i) = r_i, \tag{4}$$

be a decomposition of the domain Ω into a sequence of overlapping sub-domains Ω_i, where $r_i \leq N$ and $\Omega_i \cap \Omega_j = \Omega_{ij} \neq \emptyset$ when the subdomains are adjacent.

Associate to the decomposition (4), we give the following:

Definition 2 (*The restriction and the extension operator*) If $\mathbf{w} = (w_k)_{k\in\Omega} \in \Re^N$ then

$$RO_i(\mathbf{w}) := (w_k)_{k\in\Omega_i} \in \Re^{r_i} \qquad \forall i = 1, \ldots, p$$

is the restriction operator acting on $\mathbf{w}$. In the same way, if $\mathbf{z} = (z_k)_{k\in\Omega_i}$, then it is

$$EO_i(\mathbf{z}) := (\tilde{z}_k)_{k\in\Omega} \in \Re^N \qquad \forall i = 1, \ldots, p$$

where

$$\tilde{z}_k := \begin{cases} z_k & k \in \Omega_i \\ 0 & elsewhere \end{cases} \tag{5}$$

is the extension operator acting on $\mathbf{z}$.

We shall use the notations $RO_i(\mathbf{w}) \equiv \mathbf{w}^{RO_i}$ and $EO_i(\mathbf{z}) \equiv \mathbf{z}^{EO_i}$.

Remark For any vector $\mathbf{w} \in \Re^N$, associated to the domain decomposition (4), it results that

$$\mathbf{w} = \sum_{i=1,p} EO_i\left[\mathbf{w}^{RO_i}\right]. \tag{6}$$

Given p vectors $\mathbf{w}^{EO_i} \in \Re^{r_i}$, the vector summation

$$\mathbf{w} := \sum_{i=1,p} \mathbf{w}^{EO_i} \tag{7}$$

is such that, for any $j \in \Omega$:

$$RO_j[\mathbf{w}] = RO_j\left[\sum_{i=1,p} \mathbf{w}^{EO_i}\right] = \mathbf{w}^{RO_j}$$

Definition 3 (*The functional restriction operator*) Let

$$J : \mathbf{w} \in \Re^N \mapsto J(\mathbf{w}) \in \Re$$

be an operator defined in $\Re^N$, then we generalize the definition of the restriction operator RO_i acting on J, as follows:

$$RO_i[J] : J \mapsto RO_i[J] \qquad \forall i = 1, \ldots, p \tag{8}$$

where $RO_i[J]$ is defined as follows:

$$RO_i[J] : \mathbf{w} \in \Re^N \mapsto \begin{cases} J\left(\mathbf{w}^{RO_k}\right), & k \in \Omega_i \wedge \forall\, j \neq k,\ k \notin \Omega_j, \\ \frac{1}{2} J\left(\mathbf{w}^{RO_k}\right), & \exists\, j\ : k \in \Omega_i \cap \Omega_j \end{cases}$$

For simplicity of notations, and also for underlining that the restriction operator is associated to the domain decomposition (4) we pose:

$$RO_i[J] \equiv J_{\Omega_i}.$$

Definition 4 (*The functional extension operator*) We generalize the definition of the extension operator EO_i acting on J_{Ω_i} as

$$EO_i[J_{\Omega_i}] : J_{\Omega_i} \mapsto EO_i[J_{\Omega_i}],$$

where

$$EO_i[J_{\Omega_i}] : \mathbf{w} \in \Re^N \mapsto \begin{cases} J\left(EO_i\left(\mathbf{w}^{RO_k}\right)\right) & k \in \Omega_i \\ 0 & k \notin \Omega_i \end{cases}. \tag{9}$$

We note that the (9) can be written as

$$J_{\Omega_i}^{EO_i}(\mathbf{w}) = EO_i[J_{\Omega_i}](\mathbf{w}) = J(EO_i(RO_i[\mathbf{w}])) \tag{10}$$

Proposition 1 *(Functional Decomposition) Let $\Omega = \bigcup_{i=1}^{p} \Omega_i$ be a domain decomposition as defined in (4) and let J be a functional defined in $\Re^N$. It holds the following decomposition of J:*

$$J \equiv \sum_{i=1,p} J_{\Omega_i}^{EO_i}, \tag{11}$$

where

$$J_{\Omega_i} : \Re^{r_i} \mapsto \Re.$$

Proof From the (10) it follows that, if $\mathbf{w} \in \Re^N$, then

$$\sum_{i=1,p} J_{\Omega_i}^{EO_i}(\mathbf{w}) = \sum_{i=1,p} EO_i\left[J_{\Omega_i}\right](\mathbf{w}) = \sum_{i=1,p} \left[J_{\Omega_i}\left(RO_i\left[\mathbf{w}\right]\right)\right]^{EO_i}. \tag{12}$$

From the (6), (9) and (12) it results that

$$\sum_{i=1,p}\left[J_{\Omega_i}\left(RO_i[\mathbf{w}]\right)\right]^{EO_i} = \sum_{i=1,p} J\left(EO_i\left(\left(\mathbf{w}^{RO_i}\right)\right)\right)$$
$$= J\left[\sum_{i=1,p}\left(\mathbf{w}^{RO_i}\right)^{EO_i}\right] = J(\mathbf{w}) \ . \qquad (13)$$

From the (12) and (13), the (11) follows. □

3 The TR Functional Decomposition

Let:

$$J_{TR,\Omega}(\mathbf{w},\lambda) := \|T\mathbf{w}-\mathbf{b}\|_2 + \lambda\|Q\mathbf{w}\|_2 \qquad (14)$$

be the TR functional defined on Ω, and let:

$$\mathbf{w}_\lambda^{TR} = argmin_{\mathbf{w}\in\Re^N} J_{TR,\Omega}(\mathbf{w},\lambda) \qquad (15)$$

be its solution.
We now introduce the *local* TR functionals which describes the local problems on each sub-domain Ω_i.

Definition 5 (*Local TR problem*) Let $\Omega = \bigcup_{i=1}^{p}\Omega_i$ be a domain decomposition as defined in (4). For any vector $\mathbf{w} \in \Re^N$, let: $J_{TR,\Omega_i} = RO_i[J_{TR}]$ be an operator as defined in (8) and let

$$\tilde{J}_{\Omega_{ij}} : \mathbf{w}^{RO_{ij}} \mapsto \tilde{J}_{\Omega_{ij}}(\mathbf{w}^{RO_{ij}}) \in \Re$$

be a quadratic operator defined in $\Re^{card(\Omega_{ij})}$. The operator

$$J_{LTR,\Omega_i}(\mathbf{w}^{RO_i},\lambda_i,\omega_i) := J_{TR,\Omega_i}(\mathbf{w}^{RO_i},\lambda_i) + \omega_i\tilde{J}_{\Omega_{ij}}(\mathbf{w}^{RO_{ij}}) \qquad (16)$$

is the local TR functional defined on Ω_i. The parameters λ_i, ω_i are said local regularization parameters. Then

$$\mathbf{w}_{\lambda_i,\omega_i}^{TR_i} := argmin_{\mathbf{w}^{RO_i}} J_{LTR,\Omega_i}(\mathbf{w}^{RO_i},\lambda_i,\omega_i). \qquad (17)$$

is the solution of the local TR problem. Since the local TR functional is quadratic this solution is also unique, once index i has been fixed.

In practice, J_{LTR,Ω_i} is obtained from a restriction of the TR functional $J_{TR,\Omega}$ in (14), by adding a *local* functional defined on the overlapping regions in Ω_{ij}. This is in order to enforce a sufficient regularity of each solution of the local TR problems onto the overlap region between adjacent domains Ω_i and Ω_j. The operator $\tilde{J}_{\Omega_{ij}}$ can be suitably defined according to the specific requirements of the solution of the TR problem.

Remark From the (11) it follows that

$$J_{TR,\Omega} = \sum_{i=1,p} J_{LTR,\Omega_i}^{EO_i} = \underbrace{\sum_{i=1,p} J_{TR,\Omega_i}^{EO_i}}_{J_{TR,\Omega}} + \underbrace{\sum_{i=1,p} \omega_i \tilde{J}_{\Omega_{ij}}^{EO_i}}_{\mathcal{C}} . \tag{18}$$

where $\mathcal{C}$ is the convex functional which has to be defined in order to guarantee a sufficient regularity of each solution of the local problems onto the overlap region between adjacent domains Ω_i and Ω_j.

The following result relates the solution of TR problem in (15) to the solutions of the local TR problems in (16). From simplicity of notations, in the following theorem we assume that $\mathcal{C}$ is quadratic functional. The result still holds if this is a more general convex functional as in [1,4,9,10,18].

Theorem 1 *Let*

$$\Omega = \bigcup_{i=1,p} \Omega_i$$

be a domain decomposition of Ω defined in (4), and let (18) be the associated functional decomposition. Then let $\mathbf{w}_\lambda^{TR}$ be defined in (15) and let $\widehat{\mathbf{w}}_\lambda^{TR}$ be defined as follows:

$$\widehat{\mathbf{w}}_\lambda^{TR} = \sum_i \left(\mathbf{w}_{\lambda_i,\omega_i}^{TR_i}\right)^{EO_i} .$$

It is:

$$\widehat{\mathbf{w}}_\lambda^{TR} = \mathbf{w}_\lambda^{TR} .$$

Proof The functional $J_{TR,\Omega}$ is convex, as well as all the functionals J_{LTR,Ω_i}, so their (unique) minimum, $\mathbf{w}_\lambda^{TR}$ and $\mathbf{w}_{\lambda_i,\omega_i}^{TR_i}$, respectively, are obtained as zero of their gradients, i.e.:

$$\nabla J_{TR,\Omega}\left[\mathbf{w}_\lambda^{TR}\right] = 0, \quad \nabla J_{LTR,\Omega_i}\left[\mathbf{w}_{\lambda_i,\omega_i}^{TR_i}\right] = 0 . \tag{19}$$

From (16) it follows that

$$\nabla J_{LTR,\Omega_i}\left[\mathbf{w}_{\lambda_i,\omega_i}^{TR_i}\right] = \nabla J_{TR,\Omega_i}\left[\mathbf{w}_{\lambda_i,\omega_i}^{TR_i}\right]. \tag{20}$$

From (6) it is:

$$\mathbf{w}_{\lambda_i,\omega_i}^{TR_i} = \sum_{j=1,p} \left(\mathbf{w}_{\lambda_j,\omega_i}^{TR_j}\right)^{EO_j} , \quad on \quad \Omega_i . \tag{21}$$

From (19), (20) and (21), it follows that

$$0 = \nabla J_{TR,\Omega_i}\left(\mathbf{w}_{\lambda_i,\omega_i}^{TR_i}\right) = \nabla J_{TR,\Omega_i}^{EO_i}\left(\sum_{j=1,p}\left(\mathbf{w}_{\lambda_i,\omega_i}^{EO_i}\right)^{TR_i}\right). \tag{22}$$

By summing each equation in (22) for $i = 1, \ldots, p$ on all sub-domains Ω_i, from (22) it follows that:

$$\sum_i \nabla J_{TR,\Omega_i}^{EO_i}\left(\sum_j\left(\mathbf{w}_{\lambda_j,\omega_j}^{EO_j}\right)^{TR}\right) = 0 \Leftrightarrow \sum_i \nabla J_{LTR,\Omega_i}^{EO_i}\left(\hat{\mathbf{w}}^{TR}\right) = 0. \tag{23}$$

From the linearity of the gradients of J_{TR,Ω_i}, it is

$$\sum_i \nabla J_{TR,\Omega_i}^{EO_i}\left(\hat{\mathbf{w}}_\lambda^{TR}\right) = \nabla \sum_i J_{TR,\Omega_i}^{EO_i}\left(\hat{\mathbf{w}}_\lambda^{TR}\right) = \nabla J\left(\hat{\mathbf{w}}_\lambda^{TR}\right) . \tag{24}$$

Hence, from (24) it follows

$$\sum_i \nabla J_{TR,\Omega_i}^{EO_i}\left(\hat{\mathbf{w}}_\lambda^{TR}\right) = 0 \Leftrightarrow \nabla J\left(\hat{\mathbf{w}}_\lambda^{TR}\right) = 0 .$$

Finally,

$$\nabla J\left(\widehat{\mathbf{w}}_\lambda^{TR}\right) = 0 \Rightarrow \widehat{\mathbf{w}}_\lambda^{TR} \equiv \mathbf{w}_\lambda^{TR} ,$$

where the last equality holds because the minimum is unique. □

This result says that $\mathbf{w}_\lambda^{TR}$, the minimum of $J_{TR,\Omega}$, can be regarded as a piecewise function obtained by patching together the minimum of the operators J_{LTR,Ω_i}, $\mathbf{w}_{\lambda_\mathbf{i},\omega_\mathbf{i}}{}^{TR_i}$, i.e. by using the domain decomposition, the global minimum of the operator J can be obtained by patching together the minimum functions of the *local* functionals J_{LTR,Ω_i}. As it will be explained in the next section, this result has important implications from the computational viewpoint.

3.1 The Case Study: The Data Assimilation Inverse Problem

Let $t \in [0, T]$ denote the time variable. Let $\Delta t > 0$ and :

$$\mathcal{M} : \begin{cases} u^{true}(t-\Delta t, x) \mapsto u^{true}(t, x), \ \forall\, (t, x) \in [0, T] \times \Omega \\ u(t_0, x) = u_0(x), \qquad\qquad\qquad t_0 = 0, \quad x \in \Omega \end{cases} \tag{25}$$

be a symbolic description of a predictive 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 [21].
Given

- $\mathcal{D} = \{x_j\}_{j=1,\dots,N} \in \Re^N$ where $x_j \in \Re^3$, a discretization of a physical domain;
- $\mathbf{M}$: a discretization of $\mathcal{M}$;
- $\mathbf{u}_0^{\mathcal{M}} = \{u_0^j\}_{j=1,\dots,N}^{\mathcal{M}} \equiv \{u(t_0, x_j)\}_{j=1,\dots,N}^{\mathcal{M}} \in \Re^N$: numerical solution of $\mathcal{M}$ on $\mathcal{D}$. This is the so called background estimates, i.e. the initial states at time t_0; it is assumed to be known, usually provided by a previous forecast.
- $\mathbf{u}^{\mathbf{M}} = \{u^j\}_{j=1,\dots,N} \equiv \{u(x_j)\}_{j=1,\dots,N} \in \Re^N$: numerical solution of $\mathbf{M}$ on $\mathcal{D}$;
- $\mathbf{u}^{true} = \{u(x_j)^{true}\}_{j=1,\dots,N}$: the vector values of the reference solution of $\mathcal{M}$ computed on $\mathcal{D}$ at t fixed;
- $\mathbf{v} = \{v(y_j)\}_{j=1,\dots,nobs}$: the vector values of the observations on $\mathcal{D}$ at a t fixed;
- $\mathbf{H} \in \Re^{N\times nobs}$ is the matrix obtained by the approximation of the Jacobian of $\mathcal{H}$ and $nobs << N$;
- $\mathbf{R} = \sigma_o^2\mathbf{I}$, and $\mathbf{B} = \sigma_B^2\mathbf{C}_B$ the covariance matrices of the errors on the observations $\mathbf{v}$ and on the system state $\mathbf{u}^{\mathbf{M}}$, respectively. These matrices are symmetric and positive definite.

Definition 6 (*The DA inverse problem*) Let $\mathbf{M} \in \Re^{N\times N}$ and $\mathbf{H} \in \Re^{nobs\times N}$ linear operators, and the vector $\mathbf{v} \in \Re^{nobs\times 1}$, where $N >> nobs$, the inverse problem concerning the computation of $\mathbf{u} : \Omega \mapsto \Re^{N\times 1}$ such that

$$\mathbf{v} = \mathbf{H}\mathbf{u} \tag{26}$$

subject to the constraint $\mathbf{u} = \mathbf{u}^{\mathcal{M}}$ where $\mathbf{u}^{\mathcal{M}} = \mathbf{M}[\mathbf{u}]$ is an inverse problem plus constraint.

If (26) is ill posed, the TR formulation provides the approximation $\mathbf{u}(\lambda)$ of $\mathbf{u}$, where λ is the regularization parameter, as stated in the following

Definition 7 (*The DA-TR problem*) To compute:

$$\mathbf{u}(\lambda) = argmin_{\mathbf{u}} J(\mathbf{u}) = argmin_u \left\{ \|\mathbf{H}\mathbf{u} - \mathbf{v}\|_{\mathbf{R}}^2 + \lambda \|\mathbf{u} - \mathbf{u}^{\mathcal{M}}\|_{\mathbf{B}}^2 \right\} \tag{27}$$

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.

The relation between the DA-TR formulation in (27) and the standard TR formulation presented in (3) is detailed in the following

Proposition 2 *If*

$$\mathbf{T} = \mathbf{H}\mathbf{C}_B^{1/2}$$

and

$$\mathbf{b} = \mathbf{v} - \mathbf{H}\mathbf{u}^{\mathcal{M}}$$

then the DA-TR problem defined in (27) is equivalent to the TR formulation given in (3), where

$$\mathbf{w} = \mathbf{C}_B^{-1/2}(\mathbf{u} - \mathbf{u}^{\mathcal{M}})$$

Proof It is straightforward. □

4 Scalability Metrics of the D-TR Based Algorithm

In this section we introduce the metrics we shall use for evaluating the performance of a parallel algorithm. Perhaps the simplest of these metrics is the elapsed time taken to solve a given problem on a given parallel platform. However, this measure suffers from some drawbacks. Namely, the execution time of a parallel algorithm depends not only on input size but also on the number of processing elements used, and their relative computation and interprocess communication speeds. Furthermore, many factors contribute to the scalability, including the architecture of the parallel computer and the parallel implementation of the algorithm. For these reasons, we focus on metrics for quantifying the performance and the scalability of the algorithm itself. We first give the following

Definition 8 (*TR-Algorithm*) Let $A(\Omega)$ be the algorithm solving the TR problem in (15) defined in Ω.

Definition 9 (*D-TR algorithm*) If $p \in N$, and $p > 1$, the algorithm which solves the TR problem in (15) by solving the local TR problems in (16) associated to the decomposition given in (4), is

$$\mathcal{A}_p(\Omega) := \{\mathcal{A}(\Omega_1), \mathcal{A}(\Omega_2), \ldots, \mathcal{A}(\Omega_p)\} \tag{28}$$

where $\mathcal{A}(\Omega_i)$ is the local algorithm solving the local TR problem (16) on Ω_i.

$\mathcal{A}_p(\Omega)$ is said the D-TR algorithm solving the problem in (16) associated to the decomposition given in (4).
We introduce the symbol $T(N)$ for denoting the time complexity of the algorithm $\mathcal{A}(\Omega)$; in the same way the symbol $T(r_i)$ denotes the time complexity of $\mathcal{A}(\Omega_i)$, where $r_i = card(\Omega_i)$.
In order to quantify the scalability of the D-TR algorithm $\mathcal{A}_p(\Omega)$ we use the following measure [7]

Definition 10 (*The D-TR algorithm scale up factor*) Let $p \in N$ and $p > 1$. $\forall\, i \neq j$ we define the (relative) scale up factor of $\mathcal{A}_p(\Omega)$, in going from 1 to p, the following ratio:

$$Sc^{f}_{1,p}(\mathcal{A}_p(\Omega)) = \frac{T(N)}{p \cdot T(r_i)} \quad . \tag{29}$$

The scale up factor quantify the benefit of the decomposition, i.e., how much the time complexity of the algorithm $\mathcal{A}(\Omega)$ may be reduced with respect to the D-TR based algorithm $\mathcal{A}_p(\Omega)$.
Let $SW(\mathcal{A}(\Omega), nproc)$ denote the software implementing $\mathcal{A}(\Omega)$ on a fixed processing architecture made of $nproc$ processing elements. We recall the following

Definition 11 (*Floating point execution time*) Let t_{flop} denote the unitary time required for the execution in Processing Element (PE) of one floating point operation. The execution time needed to $SW(\mathcal{A}(\Omega), nproc)$ for performing $T(N)$ floating point operations, is

$$T_{flop}(N) = T(N) \times t_{flop}. \tag{30}$$

In addition to performing essential computation (i.e., computation that would be performed by the serial program for solving the same problem instance), a parallel software may also spend time in interprocess communication, idling, and excess computation (computation not performed by the serial formulation). We will denote this time as the software elapsed execution time.

Definition 12 (*Software elapsed execution time*)

$$T^{nproc}(N) := T^{nproc}_{flop}(N) + T^{nproc}_{oh}(N) \tag{31}$$

denote the software elapsed execution time of $SW(\mathcal{A}(\Omega, nproc))$ given by time for computation plus an overhead which is given by synchronization, memory accesses and communication time also (see [17] for details).

Clearly, from the (30) and the (31) if $nproc = 1$ it follows that

$$T_{flop}(N) = T^{1}_{flop}(N)$$

By using the (31), in the next proposition we express the measured scale up factor in terms of the local speed up. To this end we need to introduce the following quantity

Definition 13 (*Local speed up factor*) The ratio:

$$s^{loc}_{nproc}(\mathcal{A}(\Omega_i)) = \frac{T_{flop}(r_i)}{T^{nproc}(r_i)}$$

denotes the speed up of the (local) algorithm $\mathcal{A}(\Omega_i)$.

Definition 14 (*Surface-to-volume ratio*) Let $\frac{S}{V}$ 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). For $\mathcal{A}(\Omega_i)$ this is

$$\frac{S}{V}(\mathcal{A}(\Omega_i)) := \frac{T_{oh}(r_i)}{T_{flop}(r_i)}. \tag{32}$$

Definition 15 (*Measured software scale-up*) Let

$$Sc_{1,p}^{meas}(\mathcal{A}_p(\Omega)) := \frac{T_{flop}(N)}{p \cdot (T_{flop}(r_i) + T_{oh}(r_i))}. \tag{33}$$

be the measured software scale-up in going from 1 to p.

We prove the following result relating the measured scale-up and scale-up factor under the condition that $r_i = \frac{N}{p} = r(= constant)$, for $i = 1, \dots, p$.

Proposition 3 *Let us consider an uniform decomposition, that is $r_i = \frac{N}{p} = r(= constant)$, for $i = 1, \dots, p$. Then, it holds that*

$$Sc_{1,p}^{meas}(\mathcal{A}_p(\Omega)) = \alpha Sc_{1,p}^{f}(\mathcal{A}_p(\Omega)) \tag{34}$$

with

$$\alpha := \frac{s_{nproc}^{loc}(A(\Omega_i))}{1 + s_{nproc}^{loc}(A(\Omega_i))\frac{S}{V}} \tag{35}$$

Proof

$$Sc_{1,p}^{meas}(\mathcal{A}_p(\Omega)) = \frac{T_{flop}(N)}{\frac{pT^{nproc}(r)}{s_{nproc}^{loc}(A(\Omega_i))} + pT_{oh}(r)} = \frac{s_{nproc}^{loc}(A(\Omega_i))\frac{T_{flop}(N)}{pT^{nproc}(r)}}{1 + \frac{s_{nproc}^{loc}(A(\Omega_i))T_{oh}(r)}{T_{flop}(r)}}. \tag{36}$$

If

$$\alpha := \frac{s_{nproc}^{loc}(A(\Omega_i))}{1 + \frac{s_{nproc}^{loc}(A(\Omega_i))T_{oh}(r)}{T_{flop}(r)}} = \frac{s_{nproc}^{loc}(A(\Omega_i))}{1 + s_{nproc}^{loc}(A(\Omega_i))\frac{S}{V}}$$

from (36) it comes the (34). □

Finally, last proposition allows us to examine the benefit on the measured scale up arising from the speed up of the local parallel algorithm, mainly in the presence of a multilevel decomposition, where $s_{nproc}^{loc} > 1$.

Proposition 4 *Under the same hypothesis of Proposition 3, if*

$$0 \le \frac{S}{V} < 1 - \frac{1}{s_{nproc}^{loc}(A(\Omega_i))}, \tag{37}$$

it holds that

$$s_{nproc}^{loc} \in [1, p] \Rightarrow Sc_{1,p}^{meas} \in]Sc_{1,p}^{f}, p \cdot Sc_{1,p}^{p}[.$$

Proof From (37) it follows

- if $s_{nproc}^{loc} = 1$ then $\alpha < 1 \Leftrightarrow Sc_{1,p}^{meas} < Sc_{1,p}^{f}$;
- if $s_{nproc}^{loc} > 1$ then $\alpha > 1 \Leftrightarrow Sc_{1,p}^{meas} > Sc_{1,p}^{f}$;
- if $s_{nproc}^{loc} = p$ then $1 < \alpha < p \Rightarrow Sc_{1,p}^{meas} < p \cdot Sc_{1,p}^{f}$.

□

The next result allows to further detail the behaviour of the scale-up factor [7]

Proposition 5 *Let us consider an uniform decomposition, that is* $r_i = \frac{N}{p} = r(= constant)$, *for* $i = 1, \ldots, p$. *Let*

$$T(N) = a_d N^d + a_{d-1} N^{d-1} + \ldots + a_0, \quad a_d \neq 0$$

be a polynomial of degree $d > 1$, *then scale up factor is*

$$Sc_{1,p}^{f}(N) = \frac{T(N)}{p \cdot T(r)} = \alpha(N, p)\, p^{d-1} \tag{38}$$

where

$$\alpha(N, p) = \frac{a_d + a_{d-1}\frac{1}{N} + \ldots + \frac{a_0}{N^d}}{a_d + a_{d-1}\frac{p}{N} + \ldots + \frac{a_0 p^d}{N^d}}.$$

Corollary 1 *Under the same hypothesis of Proposition 5, if* $a_i = 0 \quad \forall i \in [0, d-1]$, *then* $\beta = 1$, *i.e.*

$$p = N \Rightarrow \alpha(N, p) = 1$$

Finally

$$\lim_{N\to\infty} \alpha(N, p) = 1.$$

Corollary 2 *Under the same hypothesis of Proposition 5, if N is fixed, it is*

$$p = N \Rightarrow Sc_{1,p}^{f}(\mathcal{A}_p(\Omega)) = \beta \cdot N^{d-1};$$

while, if p is fixed

$$\lim_{N\to\infty} Sc_{1,p}^{f}(\mathcal{A}_p(\Omega)) = const \neq 0.$$

In conclusion, we may analyse the algorithm scalability by looking at the behaviour of the parameter $\alpha(N, p)$ in terms of N and P. For instance, we observe that

1. if p increases and N is fixed, the scale up factor increases but the surface-to-volume ratio increases since N/p - i.e. the size of each sub-domain - decreases.
2. if p is fixed and N increases, the scale up factor stagnate and the surface-to-volume ration decreases because N/p increases.
3. if N and p increase such that N/p increases, the scale up factor increases and the surface-to-volume ratio decreases.

This means that in cases 1) and 2), i.e. for a fixed size application or a fixed number of sub domains, the algorithm has a poor scalability, because it needs to find the appropriate value of the number of sub domains, p, giving the right trade off between the scale up and the overhead of the algorithm. On the other hand, in case 3), i.e. for large scale problems the algorithm has a good scalability.

5 Test Cases: A Multilevel Hybrid Parallel D-TR Based Algorithm for the Data Assimilation

Basically, there exist three approaches to design parallel algorithms: the first strategy corresponds to the standard fine-grained concurrency of the floating-point operations (it exploits concurrency inside the parts that represent the main computational bottlenecks on the way to gain performance of the whole algorithm), the second one is based on the problem decomposition and introduces concurrency at a coarser level of computation, and finally, the last one combines the previous two strategies. The main difference between first two approaches and the last one is essentially due to the different amounts of parallelism that could be extracted from a given computational problem. Moreover, the appropriate mapping depends upon both the specification of the algorithm and the underlying architecture. Indeed, the first strategy is best suited for extracting the concurrency of internode distributed memory multiprocessors, such as custom multiprocessors with a low-latency network connection (i.e. the nodes in a single system communicating among them by means of dedicated fast networks or high performance switches as well as graphic accelerators, sharing resources in a single CPU). The second strategy is oriented for exploiting the intranode parallelism of distributed memory multiprocessors (i.e. several systems connected among them by geographic networks) because it does not require a strong cooperation among computing elements, and the last one should be more adapted on hybrid architectures, where parallelism is searched both at the finest and coarsest levels. As will be described below, in order to better exploit the emerging architectures performance, we implement the third strategy by combining Algorithm 1 (designed for implementing the coarsest level of parallelism) and Algorithm 2 (designed for implementing the finest level of parallelism) on a testbed computing environment.

5.1 The Testbed Computing Environment

Algorithm 1 $\mathcal{A}_p(\Omega)$: D-TR algorithm

```
1: procedure D- TR($in$ : $\mathbf{b}$, $\mathbf{T}$, $\mathbf{Q}$, $out$ : $\mathbf{w}$)
2:     Data Decomposition among $nproc$ processing elements
3:     for i=1, nproc
4:     $k := 0$, $\mathbf{w}_i^0 = \mathbf{0}$, $\mathbf{w}_{ij} = \mathbf{0}$
5:     repeat
6:         $k := k + 1$
7:         Call Procedure Local -TR($in$ : $\mathbf{Q_i}$, $\mathbf{b}_i$, $\mathbf{T}_i$, $\mathbf{w}_i^{k-1}$, $\mathbf{w}_{ij}^{k-1}$; $out$ : $\mathbf{w}_i^k$)
8:         Exchange $\mathbf{w}_{ij}^k$ between adjacent subdomains
9:     until $\|\mathbf{w}_i^k - \mathbf{w}_i^{k-1}\| < TOL$
10:     end for
11:     Gather of $\mathbf{w}_i^k$ : $\mathbf{w} = \sum_i \mathbf{w}_i^k$
12: end procedure
```

Algorithm 2 $\mathcal{A}(\Omega_i)$: Local TR algorithm

```
1: procedure LOCAL-TR($in$ : $\mathbf{Q_i}$, $\mathbf{b}_i$, $\mathbf{T}_i$, $\tilde{\mathbf{w}}_i$, $\mathbf{w}_{ij}$ ; $out$ : $\bar{\mathbf{w}}_i$)
2:     Compute $J_{\Omega_i}$ on $\Omega_i$
3:     Compute $\bar{\mathbf{w}}_i = argmin\left\{J_{\Omega_i}(\mathbf{w}_i)\right\}$
4: end procedure
```

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

- PE_1 (for the coarsest level of granularity): a *Multiple-Instruction, Multiple-Data* (MIMD) architecture made of 8 nodes which consist of distributed memory DELL M600 blades connected by a 10 Gigabit Ethernet technology. Each blade consists of 2 Intel Xeon@2.33GHz quadcore processors sharing the same local 16 GB RAM memory for a total of 8 cores per blade and of 64 total cores.
- PE_2 (for the finest level of granularity): a Kepler architecture of the GK110 GPU [28], which consists of a set of 13 programmable *Single-Instruction, Multiple-Data* (SIMD) *Streaming Multiprocessors* (SMXs), connected to a quad-core Intel i7 CPU running at 3.07GHz, 12 GB of RAM. For host(CPU)-to-device(GPU) memory transfers CUDA enabled graphic cards are connect to a PC motherboard via a PCI-Express (PCIe) BUS [32].

The D-TR algorithm $\mathcal{A}_p(\Omega)$ which solves the TR problem (see (15)) on PE_1 consists of copies of the local TR algorithms $\mathcal{A}(\Omega_i)$ (see (17)) assigned to the PE_2 architecture.

5.2 Experiments Set up

Step 2 of Algorithm 1 and the Steps 2-3 of Algorithm 2 depend on the TR mathematical problem. For our experiments, we use the Data Assimilation problem described in Sect. 3.1, and the computational model:

$$J_{\Omega_i}(\mathbf{w}_i, \lambda_i, \omega_i) = J_{TR,\Omega_i}(\mathbf{w}_i, \lambda_i) + \tilde{J}_{\Omega_{ij}}, \tag{39}$$

where $J_{TR,\Omega_i}(\mathbf{w}_i, \lambda_i)$ is the function $J(\mathbf{u})$ defined in (27) (see also Proposition 2), and

$$\tilde{J}_{\Omega_{ij}} = \omega_i \|\mathbf{w}_{ij} - \mathbf{w}_{ji}\|^2_{\mathbf{B_{ij}}},$$

where $\mathbf{w}_{ij} = \mathbf{w}_i/\Omega_{ij}$ and $\mathbf{w}_{ji} = \mathbf{w}_j/\Omega_{ij}$.
More details about the parameters for setting-up this Oceanographic DA model are provided in [8,9].
The data decomposition (Step 2-Algorithm 1) strategy we adopted, uses the following correspondence between p and $nproc$:

$$p \leftrightarrow nproc$$

which means that the number of subdomains coincides with the number of available processors on PE_1.
According to the characteristics of the physical domain in a shallow water model equation we are considering, we have $N = n_x \times n_y \times n_z$ and $n_x = n_y = n$ while $n_z = 1$. Since the unknown vectors are the fluid height or depth, and the two-dimensional fluid velocity fields, the problem size is $N = 3n^2$. We introduce a 2D uniform domain decomposition along the (x, y)-axis, such that

$$p := s \times q$$

and Ω_i such that, $\forall i = 1, \ldots, p$, $card(\Omega_i) = r$ and

$$r = nloc_x \times nloc_y \times nloc_z \tag{40}$$

with

$$nloc_x := \frac{n_x}{s} + 2o_x \;, \; nloc_y := \frac{n_y}{q} + 2o_y \;, \; nloc_z := n_z \;, \tag{41}$$

which include the overlapping $2o_x$ and $2o_y$.
Since the GPU (PE_2) can only process the data in its global memory, in a generic parallel algorithm execution, the host acquires these input data and sends them to the device memory, which concurrently calculates the minimization of the function J. To avoid continuous relatively slow data transfer from the host to the device and in order to reduce the overhead, it was decided to store the device with the entire work data prior to any processing. Namely, since for running Algorithm 2, it needs to store one matrix of size $N \times N$, two matrix of size $N \times K$, two vector of size N and one of size N

Table 1 The amount of space used for data stored by Algorithm 2 for solving a problem of size $3 \times n_{loc}^2$ on PE_2 expressed in Mbyte

n_{loc}	32	40	48	56	64	72	80	88
$Data_{mem}(Mbyte)$	177	286	485	812	1313	2041	3057	4427

plus other work space, the maximum value of $r = 3 \times n_{loc}^2$ in (40) is chosen such that the amount of data related the subdomains Ω_i (we denote it with $Data_{mem}(Mbyte)$) can be completely stored in the memory. In our case, as the global GPU memory is of 5Gbyte, we have the values of usable n described in Table 1.
As the decomposition of Ω (described in (4)) includes overlapping regions, Step 9 in Algorithm 1 implements the exchanges of boundary conditions (local data communications) between adjacent subdomains. On PE_1 we used for communications of the overlapping data, the standard Message Passing paradigm (MPI based).
Our implementation uses the matrix and vector functions in the Basic Linear Algebra Subroutines (BLAS) for PE_1 and the CUDA Basic Linear Algebra Subroutines (CUBLAS) library for PE_2. The routines used for computing the minimum of J on PE_1 and PE_2 are described in [23] and [11] respectively.

5.3 Scalability Analysis

In this section we discuss the performance of the multilevel parallel algorithm $\mathcal{A}(\Omega)$ on the reference architectures by measuring scalability in terms of *strong scaling* (which is the measure of the algorithm's capability to exploit performance of emerging computing architectures in order to minimise the time to solution for a given problem with a fixed dimension), and *weak scaling* (which is the measure of the algorithm's capability to use additional computational resources effectively to solve increasingly larger problems). We use the measured scale up described in (34), with α given in (35) and s_{nproc}^{loc} computed as values of the speed-up of the local algorithm $\mathcal{A}(\Omega_i)$ in terms of gain obtained by using the GPU (PE_2) versus the CPU (PE_1) for values of n_{loc} given in Table 2.
For the *strong scaling* analysis we fixed for $n = 64$ two configurations of the multilevel algorithm $\mathcal{A}_p(\Omega)$ for $p = 2$ and $p = 4$ as showed in Fig. 1. The value of the speed-up, for $n = 64$, is

$$s_1^{loc}(\mathcal{A}(\Omega_i)) = \frac{T_1^{PE_1}}{T_1^{PE_2}} = \frac{5389\ ms}{289\ ms} = 18.65.$$

The 2D partitioning of the computational domain requires only interactions among two adjacent subdomains. Hence, we get the surface-to-volume ratio $\frac{S}{V}$ equals to

$$\frac{S}{V} = \frac{O(r)}{O(r^3)} = O\left(\frac{1}{r^2}\right) = O\left(\frac{1}{9n^4}\right)$$

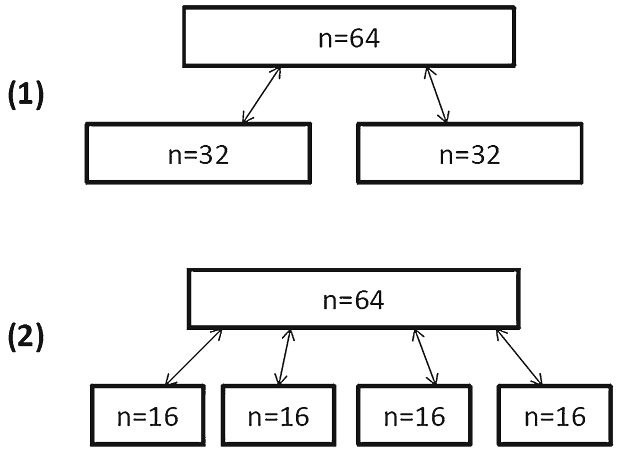

Fig. 1 Configurations of the multilevel algorithm $\mathcal{A}_p(\Omega)$ for $p = 2$ (see (1)) and $p = 4$ (see (2))

Table 2 Execution times (measured in milliseconds) of Algorithm $\mathcal{A}(\Omega_i)$ on PE_2 running for different domain size

n_{loc}	16	32	64	72	80	88
$T_D(ms)$	4	21	289	452	693	1024
$T_H(ms)$	6	98	1556	2551	4004	5606

Table 3 Values of $Sc_{1,p}^{meas}$ versus p computed with respect $n_{loc} = 16$

p	1	2	4	8	16
$Sc_{1,p}^{meas}$	-	6	24	96	384

so, from (35) we have

$$\alpha = \frac{18.65}{1 + \frac{18.65}{9 \cdot 4096}} = \frac{18.65}{1 + 0.0005} \simeq 18.42$$

From the (34) with $\alpha = 18.42$, we obtain the values of the measured scale up $Sc_{1,2}^{meas} = 73.67$ and $Sc_{1,4}^{meas} = 294.72$.
For the *weak scaling* analysis we considered the values in Table 2. In this case, once we have fixed $n_{loc} = 16$ we get

$$\alpha = \frac{T_H}{T_D} \simeq 1.5$$

and the values of $Sc_{1,p}^{meas}$ in Table 3. Here, T_H and T_D denote the execution time of Algorithm $\mathcal{A}(\Omega_i)$ running on the host (CPU) and on the device (GPU), respectively. We observe that we get a scalability prediction of the multilevel parallel algorithm, by extrapolating the values of the measured scale up computed at low processor counts to infer their evolution at large processor counts.

Performance results obtained confirm that the decomposition approach which we have introduced, allows us to leads to an effective reduction of the overall execution time needed to solve the global problem. Main outcome of the decomposition is that the parallel algorithm is oriented to better exploit the high performance of new architectures where concurrency is implemented both at the coarsest and finest levels of granularity, such as a distributed memory multiprocessor (MIMD) and a Graphic Processing Unit (GPU).

6 Conclusions

We introduced a decomposition of the Tikhonov Regularization (TR) functional suitably modified in order to enforce the matching of their solutions on the boundaries between adjacent sub problems. Instead of solving one larger TR problem we solved several smaller TR problems reproducing the TR problem at smaller dimensions. This approach produced a reduction of the time complexity of the algorithm solving the TR problems and a reduction of the overall execution time needed to solve the TR problem. These results are confirmed by the performance analysis we provided which is been discussed in terms of the algorithm and software scalability on a reference architecture made of a distributed memory multiprocessor (MIMD) and a Graphic Processing Unit (GPU) for an oceanographic Data Assimilation problem. Finally, we would like to underline that since the configurations of the multilevel parallel algorithm which we have considered in our experiments, can be regarded as a part of an algorithm designed for solving a larger problem, we observe that from the analysis of the values of the measured scale up, we can infer the expected performance at large processor counts, as those available in the exascale systems.

Acknowledgements This work was developed within the research activity of the H2020-MSCA-RISE-2016 NASDAC Project N. 691184. This work has been realized thanks to the use of the S.Co.P.E. computing infrastructure at the University of Naples.

References

1. 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. Lecture Notes in Computer Science, Vol. 2330 LNCS, Issue PART 2, 171–180, (2002)
2. Arcucci, R., D'Amore, L., Carracciuolo, L.: On the problem-decomposition of scalable 4D-Var Data Assimilation models, International Conference on High Performance Computing and Simulation (HPCS), pp. 589–594. ISBN 978-1-4673-7812-3, (2015)
3. Campagna, R., D'Amore, L., Murli, A.: An efficient algorithm for regularization of Laplace transform inversion in real case. J. Comput. Appl. Math. **210**(1–2), 84–98 (2007)
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 Comput. **32**(1), 67–83 (2006)
5. Chung, J., Nagy, J.G.: An efficient iterative approach for large scale separable nonlinear inverse problems. SIAM J. Sci. Comput. **31**(6), 4654–4674 (2010)
6. D'Amore, L., Mele, V., Laccetti, G., Murli, A.: Mathematical Approach to the Performance Evaluation of Matrix Multiply Algorithm. Lecture Notes in Computer Science, Vol. 9574, pp. 25–34 (2016)
7. D'Amore, L., Arcucci, R., Carracciuolo, L., Murli, A.: A scalable approach to variational data assimilation. J. Sci. Comput. **2**, 239–257 (2014)

8. D'Amore, L., Arcucci, R., Carracciuolo, L., Murli, A.: DD-oceanvar: a domain decomposition fully parallel data assimilation software in mediterranean sea. Procedia Comp Sci. **18**, 1235–1244 (2013)
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.: Towards a parallel component in a GPU-CUDA environment: a case study with the L-BFGS Harwell routine. J. Comput. Math. **93**(1), 59–76 (2015)
12. D'Amore, L., Campagna, R., Galletti, A., Marcellino, L., Murli, A.: A smoothing spline that approximates Laplace transform functions only known on measurements on the real axis. Inverse Probl. **28**(2), 37 (2012)
13. D'Amore, L., Murli, A.: Regularization of a fourier series method for the Laplace transform inversion with real data authors of document. Inverse Probl. **18**(4), 1185–1205 (2002)
14. D'Amore, L., Marcellino, L., Murli, A.: Image sequence inpainting: towards numerical software for detection and removal of local missing data via motion estimation. J. Comput. Appl. Math. **198**(2), 396–413 (2007)
15. Demmel, J.W., Higham, N.J., Schreiber, R.: Block LU Factorization. RIACS Technical Report no. **92–03**, (1992)
16. ETP4HPC Agenda, European Technology Platform for High Performance Computing. Strategic research agenda achieving HPC leadership in Europe, (2013)
17. Flatt, H.P., Kennedy, K.: Performance of parallel processors. Parallel Comput. **12**, 1–20 (1989)
18. Freitag, M.A., Nichols, N.K., Budd, C.J.: L1-regularisation for ill-posed problems in variational data assimilation. PAMM Proc. Appl. Math. Mech. **10**(1), 665–668 (2010)
19. Gallopoulos, E., Simoncini, V.: Iterative solution of multiple linear systems: In: Topping B.H.V., Papadrakaki M. (eds.) Theory, practice, parallelism, and applicationsIn Advances in Parallel and Vector Processing for Structural Mechanics, Proc. Second Intl. Conf. Computational Structures Technology, pp. 4751.Civil-Comp Press, Edinburgh (1994)
20. Hansen, P.C.: Rank Deficient and Discrete Ill-posed Problems. SIAM, Philadelphia (1998)
21. Kalnay, E.: Atmospheric Modeling, Data Assimilation and Predictability. Cambridge University Press, Cambridge (2003)
22. Laccetti, G., Lapegna, M., Mele, V., Romano, D., Murli, A.: A double adaptive algorithm for multidimensional integration on multicore based HPC systems. Int. J. of Parallel Progam. **40**(4), 397–409 (2012)
23. Liu, D.C., Nocedal, J.: On the limited memory BFGS method for large scale optimization. Math. Program. **45**, 503–528 (1989)
24. MAGMA Software www.icl.cs.utk.edu/plasma
25. Murli, A., D'Amore, L., Laccetti, G., Gregoretti, F., Oliva, G.: A multi-grained distributed implementation of the parallel block conjugate gradient algorithm. Concurrency Comput. Pract. Exp. **22**(15), 2053–2072 (2010)
26. Murli, A., Boccia, V., Carracciuolo, L., D'Amore, L., Laccetti, G., Lapegna, M.: Monitoring and migration of a PETSc-based parallel application for medical imaging in a grid computing PSE. IFIP Int. Federation Inf. Process. **239**, 421–432 (2007)
27. Nichols, N.K.: Mathematical concepts of data assimilation. In: Lahoz, W., Khattatov, B., Menard, R. (eds.) Data assimilation: Making Sense of Observations, pp. 13–40. Springer, Berlin (2010)
28. Nvidia, TESLA K20 GPU Active Accelerator (2012). Board spec. Available: http://www.nvidia.in/content/PDF/kepler/Tesla-K20-Active-BD-06499-001-v02
29. O'Leary, D.P., Simmons, J.A.: A bidiagonalization-regularization procedure for large scale discretizations of ill-posed problems. SIAM J. Sci. Stat. Comput. **2**, 474489 (1981)
30. Paige, C.C., Saunders, M.A.: LSQR: an algorithm for sparse linear equations and sparse least squares. ACM Trans. Math. Softw. **8**, 4371 (1982)
31. pARMS Software www-users.cs.umn.edu/saad/software/parms
32. PCIsig, tecnology specifications at http://pcisig.com/specifications/pciexpress/
33. PETSC Software www.mcs.anl.gov/petsc
34. PLASMA Software www.icl.cs.utk.edu/plasma

35. Reichel, L., Baglamana, J.: Decomposition methods for large linear discrete ill-posed problems. J. Comput. Appl. Math **198**(2), 333–343 (2007)
36. Tikhonov, A.N., Solution of incorrectly formulated problems and the regularization method, Dokl. Akad. Nauk. SSSR 151, : 501504 = Soviet Math. Dokl. **4**(1963), 1035–1038 (1963)