European Society of Computational Methods in Sciences, Engineering and Technology (ESCMSET)

Journal of Numerical Analysis, Industrial and Applied Mathematics (JNAIAM)
vol. 7, no. 3-4, 2012, pp. 91-105
ISSN 1790–8140

HPC computation issues of the incremental 3D variational data assimilation scheme in OceanVar software* †

L. D'Amore[1‡], R. Arcucci[2], L. Marcellino[3] and A. Murli[2]

1. Department of Mathematics and Application, University of Naples Federico II, Italy.
2. Centro Euro-Mediterraneo per i Cambiamenti Climatici (CMCC), Italy.
3. Department of Applied Sciences, University of Naples Parthenope, Italy.

Received 30 January, 2012; accepted in revised form 22 December, 2012

Abstract: The most significant features of Data Assimilation (DA) are that both the models and the observations are very large and non-linear (of order at least $O(10^8)$). Further, DA is an ill-posed inverse problem. Such properties make the numerical solution of DA very difficult so that, as stated in [19], *"solving this problem in "real-time" it is not always possible and many different approximations to the basic assimilation schemes are employed"*. Thus, the exploitation of advanced computing environments is mandatory, reducing the computational cost to a suitable turnaround time. This activity should be done according to a *co-design* methodology where software requirements drive hardware design decisions and hardware design constraints motivate changes in the software design to better fit within those constraints.
In this paper, we address high performance computation issues of the three dimensional DA scheme underlying the oceanographic 3D-VAR assimilation scheme, named OceanVAR, developed at CMCC (Centro Euro Mediterraneo per i Cambiamenti Climatici), in Italy. The aim is to develop a parallel software architecture which is able to effectively take advantage of the available high performance computing resources.

keywords: Data Assimilation, Inverse Problem, Parallel Software, Oceanography

MSC: 65Y05, 65F22, 65Z05

PACS: 02.30.Zz

1 Introduction

Over the last two decades or so, Data Assimilation has expanded into quite a mature and motivating area of research and applications (see [15, 17] and references therein). It seems clear

*This work has been carried out within the collaboration of the CMCC (Centro Euro Mediterraneo per i Cambiamenti Climatici), Italy. We are grateful to N. Pinardi and S. Dobricic to make concession for using OceanVAR.
†Published electronically December 15, 2012
‡Corresponding author. E-mail: luisa.damore@unina.it

that improvements in skill of Numerical Ocean Prediction are essentially due to following priority activities[8]:

1. the development of *ever more powerful computers*, allowing much finer numerical resolution and fewer approximations in the operational oceanic models;

2. the improved representation of small-scale physical processes (clouds, precipitation, turbulent transfers of heat, moisture, momentum, and radiation) within the models;

3. the use of more accurate methods which result in improved initial conditions for the models;

4. and the increased availability of data, especially satellite and aircraft data over the oceans and the Southern Hemisphere.

Moreover, the primary need to ensure a successful achievement of the benefits deriving from these activities is to advocate increased relationship and interactions among them.

In this paper we focus on the item 3 and in particular we address the impact of the availability of high performance computing resources on the development of effective DA software, within the *co-design* methodology's perspective. The aim of co-design is to reduce the large gap between the peak performance of supercomputers and the actual performance realized by today's applications. This architecture-application performance gap will get even wider with the increase in computing power being driven by a rapid escalation in the number of cores incorporated into a single chip. Co-design methodology depends on a bi-directional optimization of design parameters where software requirements drive hardware design decisions and hardware design constraints motivate changes in the software design to better fit within those constraints.
We consider the OceanVar software [4], used in Italy to produce forecasts of ocean currents to the Mediterranean Sea. OceanVAR software implements an oceanographic three-dimensional variational DA scheme.

We start from the existing sequential code, and in order to take into account both the software requirements and the system architecture, we introduce a *fine-to-coarse* parallelization strategy using the standard fine-grained concurrency of the floating-point operations, and the coarse-grained concurrency on the problem decomposition. This is essentially due to the different amounts of available parallelism of OceanVAR computations. This approach is suited for exploiting the multilevel parallelism that characterizes the architecture of concurrent multiprocessors composed of many-core CPUs. The fine-grained parallelism is aimed at extracting concurrency of multi-cores nodes. The coarse-grained parallelism is oriented to exploit internode concurrency of MIMD multiprocessors because it does not require a strong cooperation among processors.

The paper is organized as follows. In the next section we describe the DA scheme employed in OceanVar, from the numerical analysis point of view, in section 3 we discuss its conditioning and we analyze the benefits obtained from using the Cholesky factorization or the Truncated Singular Value Decomposition. In section 4 we analyze the OceanVAR parallel software co-design, we estimate the execution time underlining the more expensive routines in term of computational cost and we explain how we have introduced multi threading version of scientific libraries and their parallel version. In section 5 we report experiments and analyze performance results, while the section 6 concludes the paper.

2 DA cycle in OceanVar

Let t_i, $i = 0, 1, \ldots, n$ be a sequence of observation times and, for each i, let $x_i \equiv x(t_i) \in \Re^N$ be the vector denoting the state of the Mediterrean sea system at time t_i. The state vector considered in OceanVar contains the following variables:

$$x_i = [T, S, \eta, u, v]^T,$$

where T is the three-dimensional temperature field, S the three-dimensional salinity field, η the two-dimensional free surface elevation, and u,v are the total horizontal velocity components.
Let $\mathcal{M}(P)$ the mathematical model describing the evolution of the Mediterranean system. In particular, $\mathcal{M}(P)$ describes the evolution of the state vector x_i from t_i to t_{i+1}, and it can be expressed as follows:.

$$\mathcal{M}(P): \; x_{i+1} = \mathcal{L}_{i,i+1}(x_i), \quad i = 0, 1, \ldots, n-1 \tag{1}$$

Starting from the initial state $x_0 = x(t_0)$, which is assumed to be known, $\mathcal{M}(P)$ provides the state vector at time t_{i+1} using the state vector at time t_i. This operation is described by $\mathcal{L}_{i,i+1} : \Re^N \mapsto \Re^N$ which is the non-linear operator describing how the the model acts from t_i to t_{i+1}.
Moreover, let

$$y_{i+1} = \mathcal{H}(x_{i+1}) + \epsilon_{i+1} \quad i = 1, 2, \ldots, n-1 \tag{2}$$

be the observations vector at time t_{i+1}, where $y_{i+1} \equiv y(t_{i+1}) \in \Re^p$ and $\mathcal{H} : \Re^N \mapsto \Re^p$ is the non-linear operator collecting the observations at each time t_i.

For each $i = 1, 2, \ldots, n$, let $x_{DA}(t_i)$ be the so-called *analysis*, i.e. the estimation of the vector x_i at time t_i, obtained by using DA. OceanVAR performs DA in a sequential manner, with a time series of assimilation cycles:

For each $i = 1, 2, \ldots, n-1$ *do*

Data Assimilation cycle $(i, i+1)$**,**:

1. *given* $x_{DA}(t_i)$*,*
2. *model integration (1):*
   computation of x_{i+1}*, using* $x_{DA}(t_i)$ *as initial value,*
3. *correction of* x_{i+1} *due to observations (2):*
   computation of $x_{DA}(t_{i+1})$ *as the best estimate of* (x_{i+1}, y_{i+1})*.*
4. *provide* $x_{DA}(t_{i+1})$*.*

As a new set of observations become available the cycle is repeated. This analysis is then propagated in time.

Here we are interested on how any numerical error affecting $x_{DA}(t_i)$ propagates on $x_{DA}(t_{i+1})$. To this aim we perform the Forward Error Analysis (F.E.A.):

Forward Error Analysis on one DA cycle*: Let us assume that the DA cycle is performed on a finite precision arithmetic system. Let* $\widetilde{x_{DA}}(t_i)$ *denote the numerical value of* $x_{DA}(t_i)$*.* $\widetilde{x_{DA}}(t_i)$ *is corrupted by the presence of different kind of errors, resulting from the numerical approach (discretization, round-off, ...), all these errors can be referred by* δ_i*. By the same way,* $\widetilde{x_{i+1}}$*, indicates the perturbed value of* x_{i+1}*. We get the following steps:*

1. *given* $\widetilde{x_{DA}}(t_i) = x_{DA}(t_i) + \delta_i$,
2. *model integration (1):*
   computation of $\widetilde{x_{i+1}}$, *using* $\widetilde{x_{DA}}(t_i)$ *as initial value:*

$$\widetilde{x_{i+1}} = \mathcal{L}_{i,i+1}(\widetilde{x_{DA}(t_i)})$$

   By applying the F.E.A. on (1) we have that:

$$\widetilde{x_{i+1}} = x_{i+1} + \sigma_i \tag{3}$$

   and:

$$\sigma_i = C_A\, \mu_{\mathcal{L}}\, \delta_i \tag{4}$$

   where the constant C_A *depends on the numerical algorithm used for solving (1) and* $\mu_{\mathcal{L}}$ *depends on the condition number of* $\mathcal{L}_{i,i+1}$.
3. *correction of* $\widetilde{x_{i+1}}$ *due to observations (2):*
   computation of $\widetilde{x_{DA}}(t_{i+1})$ *as the best estimate of* $(\widetilde{x_{i+1}}, y_{i+1})$.
4. *provide*

$$\widetilde{x_{DA}}(t_{i+1}) = x_{DA}(t_{i+1}) + \eta_i \tag{5}$$

   and:

$$\eta_i = C_{DA}\mu_{DA}\delta_i \tag{6}$$

   where the constant C_{DA} *depends on the numerical algorithm used by the DA scheme and* μ_{DA} *depends on the condition number of the DA operator.*

The above analysis says that the initial error δ_i on $x_{DA}(t_i)$ propagates twice: on the numerical value of the state vector trough (4), and on the DA vector analysis through (6).
Next section focuses on the amplification factors of the DA scheme in OceanVAR.

3 Conditioning of 3DVar scheme in OceanVAR

For each time step t_i, the computational kernel of the 3DVar scheme employed in OceanVAR is the following non-linear least square problem:

$$x_{DA}(t_i) = argmin_x J(x) = argmin_x \Big\{ \|\mathcal{H}(x) - y_i\|^2_{\mathbf{R}} + \|x - x_i\|^2_{\mathbf{B}} \Big\} \tag{7}$$

$\mathbf{R}$ and $\mathbf{B}$ are the covariance matrices, whose elements provide the estimate of the errors on y_i and on x_i, respectively. As well known, they have a normal (or Gaussian) distribution. Here, $\|x\|_{\mathbf{R}}$ and $\|x\|_{\mathbf{B}}$ denote the weighted euclidean norms $\|x\|_{\mathbf{R}} = x^T\mathbf{R}^{-1}x$ and $\|x\|_{\mathbf{B}} = x^T\mathbf{B}^{-1}x$.
By using the following linearization of $\mathcal{H}$:

$$\mathcal{H}(x) = \mathcal{H}(z) + H(x - z)$$

where H is the matrix obtained by the first order approximation of the Jacobian of $\mathcal{H}$, the functional J is:

$$J(x) = (Hx - y_i)R^{-1}(Hx - y_i)^T + (x - x_i)B^{-1}(x - x_i)^T. \tag{8}$$

The minimizer of (8) is obtained by requiring that the Jacobian of J is zero, $\nabla J = 0$. This gives rise to the linear system:

$$A\delta x = b \tag{9}$$

where

$$A = B^{-1} + H^T R^{-1} H, \tag{10}$$

$b = B^{-1}x_i + H^T R^{-1} y_i$ and $\delta x = x - x_i$.

Matrix A is ill conditioned [18].
If we assume that the observations y_i are direct measurements of the state variables x_i, then $H^T H$ is a diagonal matrix, where the k-th diagonal element is unity if the k-th state variable is observed and is zero otherwise. Moreover, as in [9], we fix:

$$R = e_o^2 I \tag{11}$$

where e_o is the estimated error on y_i and

$$B = e_b^2 C \tag{12}$$

where e_b is the estimated error on x_i. The matrix B is a Gaussian matrix, if we assume that it is shift invariant also, then we have that C is a Toeplitz matrix [6] whose elements are:

$$c_{ij} = exp\left(\frac{-\Delta x^2}{2L^2}|i-j|^2\right), \quad |i-j| < \frac{N}{2}$$

where L is the length-scale, Δx is the grid spacing and N is the number of grid points.
In this case, we prove the following:

***Proposition**: Let R as in (11), B as in (12) and $H^T H$ diagonal. Then, it follows that[§]:*

$$\mu_{DA} = O(e_b^{-2}\mu(C^{-1})) \tag{13}$$

Proof:

$$\mu_{DA} \equiv \mu(A) = \mu(B^{-1} + e_o^{-2} H^T H) \leq \mu(B^{-1}) + e_o^{-2} \tag{14}$$

hence, from (12) it follows that:

$$\mu_{DA} = \mu(B^{-1}) = O(e_b^{-2}\mu(C^{-1}))$$

♣
Let λ_m denote the m-th eigenvalue of C, taking into account the structure of C eigenvalues are:

$$\lambda_m = \sum_{k=0}^{N-1} c_k exp\left(\frac{-2\pi imk}{N}\right) \tag{15}$$

that is the eigenvalues also have an exponential growth. Hence, C is ill conditioned and B too.
In order to reduce the ill conditioning of B, OceanVAR applies a two-steps transformation (preconditioning and regularization):

[§]In the following we use the big O symbol which describes the asymptotic behavior of a function when its argument tends towards a particular value or infinity. More precisely, we write $f(x) = O(g(x)$ meaning that

$$\lim_{x\to\infty} \frac{f(x)}{g(x)} = C$$

where $C \neq 0$.

1. preconditioning: since $\mathbf{B}$ is symmetric and positive definite, OceanVAR computes its Cholesky factorization $\mathbf{B} = VV^T$. By this way:
$$\mu(B) = \mu(V)^{-1/2}$$

2. regularization: OceanVAR computes the Truncated SVD (TSVD) of V. If $rank(V) = N$ then:
$$V = \sum_{j=1}^{N} \mathbf{u}_j \sigma_j \mathbf{w}_j^T = \mathbf{U}\mathbf{\Sigma}\mathbf{W}^T$$
where $\mathbf{U} = [u_1, ..., u_N]$ and $\mathbf{W} = [w_1, ..., w_N]$ matrices are orthogonal and $\mathbf{\Sigma}$ is diagonal with elements the singular values of $\mathbf{V}$:
$$\sigma_1 > \sigma_2 > \cdots > \sigma_k > \cdots > \sigma_N$$
be the SVD of V. The TSVD, instead of using all the singular values of V discards those that are less than a given threshold. These last ones are assumes to be *numerically zero*. Let K denote the number of singular values not *numerically zero*, and let:
$$V_K = \mathbf{U}\mathbf{\Sigma}_K\mathbf{W}^T$$
where $\mathbf{\Sigma}_k = diag(\sigma_1, \sigma_2, \ldots, \sigma_k, 0, 0, \ldots, 0)$. Since the matrix V_K is better conditioned than V, i.e.:
$$\mu(V_K) = \frac{\sigma_1}{\sigma_K} << \frac{\sigma_1}{\sigma_N} = \mu(V)$$
V is replaced by V_K, which provides a better conditioned approximation of V (V_K is a *regularization* of V because the contribution corresponding to the smallest singular values is filtered out [10]).

Let us express these transformations on J.

Regarding the *preconditioning*, let $d = [y - H(x)]$ be the *misfit*, by setting $v = V^T\delta x$, the operator in (8) becomes:
$$J(v) = \frac{1}{2}v^T v + \frac{1}{2}(HVv - d)^T R^{-1}(HVv - d) \tag{16}$$
The minimizer of (16) is the solution to $\nabla J = 0$, which gives rise to the linear system:
$$A'v = d. \tag{17}$$
where
$$A' = I + V^T H^T R^{-1} HV \tag{18}$$
About the conditioning of the linear system in (9), we prove the following:

Proposition: *It holds that:*
$$\mu(A') == O(1 + e_o^{-2} e_b \mu(\mathbf{C}))$$

Proof: Since R can be written as in (11), from (18) it follows that:
$$\mu(A') = \mu(I + V^T H^T R^{-1} HV) \leq 1 + \mu(V^T H^T R^{-1} HV) \leq 1 + e_o^{-2}\mu(V^T H^T HV) \tag{19}$$
and
$$1 + e_o^{-2}\mu(V^T V) = 1 + e_o^{-2} e_b \mu(C)$$

♣

Hence, the linear system (9) is still ill conditioned. It needs to use some sort of regularization.

Regularization. OceanVAR computes the TSVD of V and the vector $v_K = V_K v$ in (16):

$$V_K v = \sum_{j=1}^{K} \mathbf{u}_j \sigma_j \mathbf{w}_j^T \cdot v = \sum_{j=1}^{K} \mathbf{u}_j \sigma_j \mathbf{w}_j^T \cdot v \tag{20}$$

In [5], experimentally was found that, by choosing $K = 20$, it yields the accuracy of 10^{-2} on the solution.

4 Towards the OceanVAR co-design

Over the years users of high performance systems have observed dramatic increases in peak performance (as we approach exaflop-scale systems, we are confronted by an exponential growth in parallelism). Furthermore, the expectation is that future systems will be heterogeneous with nodes composed of many-core CPUs and GPUs. Unfortunately, this growth occurs without the commensurate improvements in effective performance delivered to applications.
The Department of Energy's Exascale Computing initiative has identified *hardware/software co-design* as a central strategy to address this situation. The strategy means the integrated co-design of tools, algorithms, and architectures to enable more efficient and timely solutions to applications: rather than ask "*what kind of scientific applications can run on an exascale system after it arrives,*" this application driven design process instead asks "*what kind of system should be built to meet the needs of the most important science problems.*"
The methodology depends on a bi-directional optimization of design parameters where software requirements drive hardware design decisions and hardware design constraints motivate changes in the software design to better fit within those constraints.

Deep analysis of application requirements drives key design decisions for the overall system architecture. The analysis provides quantitative measures of application requirements and relates them back to architectural parameters such as on-chip memory, memory bandwidth requirements and interconnect requirements. Cycle accurate simulation tools enable quantification of the performance impact on the applications when they are subjected to specific hardware constraints [21].

We will focus on the structure of OceanVAR software such that it fulfills these perspectives. We have included calls to libraries of scientific computing where possible. The use of these libraries allows finer resolution making faster the code migration on machines other than the one on which the software was generated.
Native hardware architecture of OceanVAR is the NEC supercomputers [3], the programming languages are fortran 90 and fortran 77, the scientific library was Linpack (see Figure 1).

More precisely, in order to take into account both the computing requirements of each module and the system architecture, we consider a parallelization strategies that uses the standard fine-grained concurrency of the floating-point operations, and a coarse-grained concurrency, based on the problem decomposition among computational nodes and introduces concurrency at a coarser level.

This is essentially due to the different amounts of available parallelism of a given computation. For instance, even though the coarse-grained parallelism is the most simple, under this model, the type of computation that can be efficiently parallelized is limited. Moreover, we see that these strategies are best suited for exploiting the multilevel parallelism that characterizes the architecture of concurrent multiprocessors. The fine-grained is aimed at extracting concurrency of

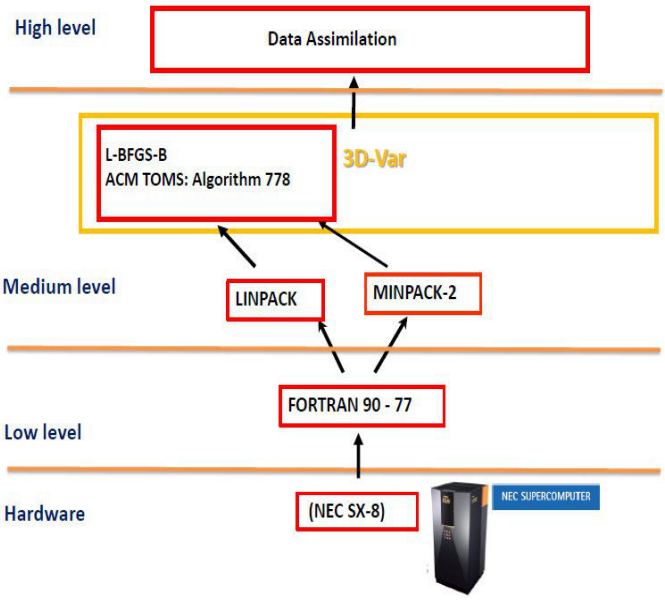

Figure 1: Software architecture of OceanVAR before parallelization

internode multiple instruction multiple data (MIMD) multiprocessors, such as custom multiprocessors with a low-latency network connection. The coarse-grained is oriented to exploit internode concurrency of MIMD multiprocessors. We feel that this is the best way to achieve the highest possible performance of a given problem on a given system configuration.

4.1 Software analysis

The approach that we choose for introducing parallelism in OceanVAR exploits concurrency inside the modules that represent the main computational bottlenecks on the way to gain performance of the whole execution. Hence, we first perform the code analysis and profiling.

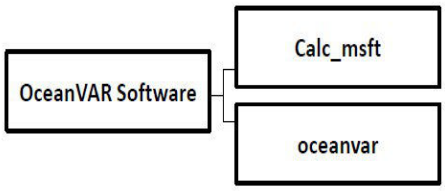

Figure 2: The two modules of OceanVAR representing the workflow of OceanVAR

The OceanVAR scheme is described in Algorithm 1.

As shown in Figure 2, the software is divided into two main modules named `Calc_msft` and `oceanvar`. `Calc_msft` module reads from the input data file values for observations. The data files

Algorithm 1 The OceanVAR scheme

1: **Acquires** the input parameters (number and size of the grid discretization, the types of observations, number of EOFs, etc.)
2: **Acquires** the observations and $\mathbf{x_M}$ vector and calculate *misfit* $\mathbf{d}$ vector
3: **Acquires** EOFs
4: **Poses** $\mathbf{v}_{num\_grid} \leftarrow \mathbf{V}^{+}_{num\_grid}\delta\bar{\mathbf{x}}_{num\_grid}$
5: $\mathbf{v}_1 = 0$
6: **for** $grid\_id \leftarrow 1, num\_grid$ **do**
7: **Calculate** quantities that define the operators $\mathbf{R^{-1}}$, $\mathbf{H}$ and $\mathbf{V}$ on $grid\_id$ grid
8: **Calculate** $J(\mathbf{v}_{grid\_id})$ e $\nabla J(\mathbf{v}_{grid\_id})$
9: **Calculate** minimum point $(\bar{\mathbf{v}})_{grid\_id}$ of $J(\mathbf{v})$ function from $\mathbf{v}_{grid\_id}$, $J(\mathbf{v}_{grid\_id})$ and $\nabla J(\mathbf{v}_{grid\_id})$ computation
10: **Calculate** $\mathbf{v}_{grid\_id+1}$ as an interpolation of $(\bar{\mathbf{v}})_{grid\_id}$
11: **end for**
12: $(\delta\bar{\mathbf{x}})_{num\_grid} \leftarrow (\mathbf{V})_{num\_grid}\mathbf{v}_{num\_grid}$
13: $\mathbf{x} \leftarrow \mathbf{x_M} + (\delta\bar{\mathbf{x}})_{num\_grid}$

used in `Calc_msft` module are of two types: ASCII and NetCDF[¶] (Network Common Data Form).

From Table 1 we note that execution time of `oceanvar` module is the most significant part of total execution time[‖]. Indeed, module `oceanvar` creates the mesh grid and calculates the minimum of the function (16). The function $J(\mathbf{v})$ is minimized using the L-BFGS minimizer [20].

	`Calc_msft`	`oceanvar`
execution time	0.16 sec.	455.10 sec.

Table 1: execution time of main modules of OceanVar

In `oceanvar` module there are two main routines: `ver_hor` and `ver_hor_ad`. These modules perform the operations $\delta\mathbf{x} = \mathbf{V}\mathbf{v}$ and $\mathbf{v} = \mathbf{V}^{+}\delta\mathbf{x}$ [4], as indicated by items 4 and 12 of Algorithm 1.
More precisely, `ver_hor` and `ver_hor_ad` provide operations between matrices, implemented using BLAS 3, and perform about 1.7277×10^{11} floating point operations.
From Table 2, we note that these routines have the greatest computational cost compared to the total execution time of the `oceanvar` module (i.e. this is about the %70).

	`ver_hor`	`ver_hor_ad`	`other`
execution time	149.16 sec.	187.66 sec.	118.27 sec.

Table 2: execution time details

Then, in the next section we describe the parallelization of these two routines, whose execution time (measured as elapsed time), from Table 2 is:

$$T(ver\_hor, ver\_hor\_ad) = 336.83$$

[¶]http://www.unidata.ucar.edu/software/netcdf/
[‖]The execution time of the routine was monitored using **etime**-function

4.2 Parallel approach

There are several different forms of parallel computing: bit-level, instruction level, data, and task parallelism. Parallel computers can be roughly classified according to the level at which the hardware supports parallelism, with multi-core and multi-processor computers having multiple processing elements within a single machine, while clusters and grids use multiple computers to work on the same task [2, 16].
The result of our work was to introduce into OceanVAR a prototype form of parallelism on two levels. Indeed, taking into account the different granularity of the computational operations we introduce concurrency during their execution. More precisely, the design of the code handled the internode communication by a standard message-passing approach (level 2), and exploited the every node capabilities of concurrency, assigning the local numerical computation to well-known optimized libraries as BLAS and LAPACK (level 1).
The introduction of (level 2)-parallelism has occurred in two routines that have the greatest computational cost compared to the total execution time of the entire application. These are: `ver_hor` and `ver_hor_ad` (step 4 and 12 of Algorithm scheme).
Within procedures `ver_hor` and `ver_hor_ad`, the quantities *free surface elevation*, *temperature* and *salinity* are calculated with different and independent operations, has therefore chosen to use an asynchronous parallelism (an "embarassingly" parallel approach). Therefore it was decided to distribute these data at three different 3 processors, each one composed by 8 cores.
Summarizing (see Figure 3), we used the multithreding version of Blas and Lapack, the standard MPI for communications and BLACS to solve linear algebra operation in parallel.

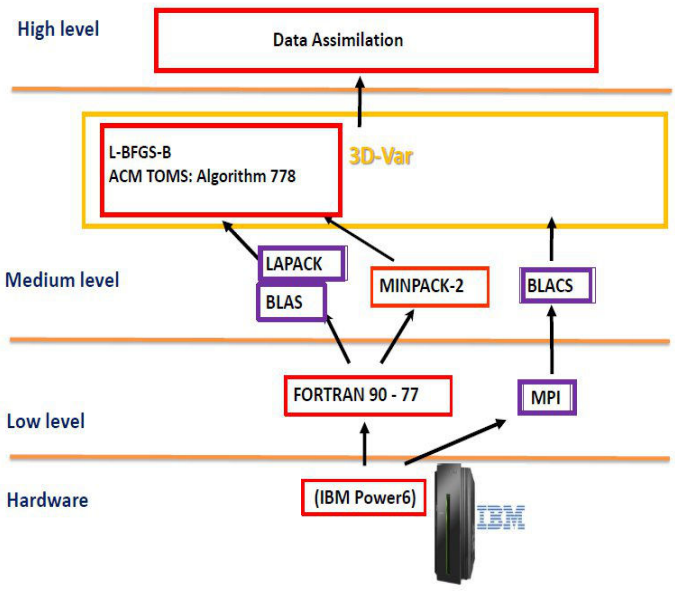

Figure 3: Software architecture of OceanVAR after parallelization

5 Experimental Results

The primary intent of this work is the exploitation of CMCC parallel computing resources in order to reduce the overall computational cost of the OceanVar code. Because of these issues, to the aim of the performance evaluation of the parallel algorithm, we consider only a real test case where the discrete domain is made of 20 matrices of size 871×253.
The computing architecture is an IBM Power6 supercomputer, consisting of 30 nodes with 16 dual-core chips. Each dual-core Power6 processor runs at 4.7 Ghz, leading to a theoretical peak of 18.8 Gflop/s per core and 601.6 Gflop/s per node (peak performance of IBM Power6).
Figure 4 shows the execution time of level 1-parallel algorithm (the plot uses a logarithmic scale). Observe that as the core number is greater than 8 the elapsed time increases, while as the core number is greater than 16, the performance degrades significantly. Indeed, the achievable performance of such an application with a fixed size, depends on deep interactions among processors, memory system and interconnected network. This means that, due to the limited amount of available parallelism, and taking into account that each node is made of 2 bi-processors each one consisting of 8 dual-core chips sharing the same local memory, there exists an "optimal" number of processors and each additional one contributes slightly less or do not. So, in our test case, this optimal configuration it that using 8 cores.

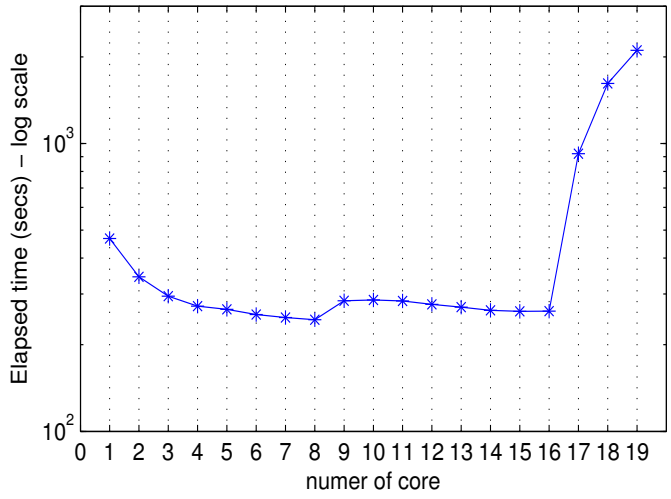

Figure 4: Execution time of level 1-parallel algorithm versus the core number.

In the following we will only refer to the elapsed time because this can be considered equals to the user time, as it is reported in Table 3.

Table 4 shows the elapsed time of `ver_hor` and `ver_hor_ad` with level 1-parallelism compared to the sequential algorithm. Note that if we focus on the problem solution in a shorter time, the performance gain is $47,29\%$, which may be acceptable.
Ayway, since we use multithreads BLAS, we expect this parallel approach to be very synchronous and do not scale with the number of cores **. This is confirmed by the performance gain measured

**Dongarra et al. [22] running a Lapack algorithm on the same architecture reaches the best efficiency (69%)

Time	sequential version	with (level 1)-parallelism.
elapsed time	455.10 sec.	244.05 sec.
user time	454.16 sec.	241.65 sec.
system time	0.95 sec.	2.40 sec.

Table 3: execution time of the oceanvar sequential version and with (level 1)-parallelism, on 8 cores

Time	sequential version	with (level 1)-parallelism.
elapsed time	336.83 sec.	123.83 sec.

Table 4: elapsed time of `ver_hor` and `ver_hor_ad`, using 8 cores

by the speed up:

$$S(8) = \frac{336.83\ sec}{123.83\ sec} = 2.72$$

and by the Gflops performed by the parallel algorithm, as shown in Figure 5.

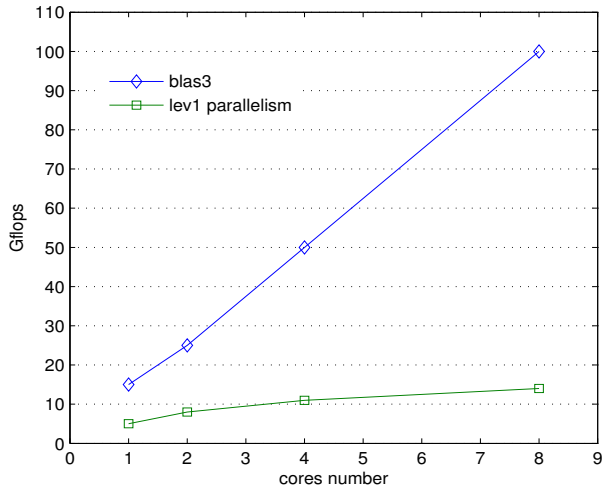

Figure 5: Gflops of of level1-parallelization algorithm and IBM power6 BLAS 3 performance, versus the core number.

Let us analyze the performance of the parallel approach obtained by introducing (levels 1 and 2)-parallelism on 3 processors (each one using 8 cores) [††].
Table 5 shows the execution time of `ver_hor` and `ver_hor_ad` by introducing (levels 1 and 2)-parallelism.

using 2 cores.

[††]as explained in the previous section the three quantities: free surface elevation, temperature and salinity are calculated with independent operations, therefore we use an asynchronous parallelism distributing these three different data at three different 3 nodes.

Time	sequential algorithm	with (level1+level2)-parallelism
elapsed time	336.83 sec.	56.31 sec.

Table 5: execution time of `ver_hor` and `ver_hor_ad` before and after (level1+level 2)-parallelization

In this case, the speed up is:

$$S(24) = \frac{336.83sec}{56.31sec} = 5.98$$

As before, by taking into account this performance metric, such gain should seem quite low if it is compared to 24 cores, anyway, this result is justified by the limited amount of available parallelism of such application code.
Let us introduce the so-called relative speed up, defined as:
Definition: *Let*

$$S' = \frac{S_{p_2}}{S_{p_1}} = \frac{\frac{T_1}{T_{p_1}}}{\frac{T_1}{T_{p_2}}}, \quad p_2 \geq p_1 \geq 1$$

where T_i denotes the execution time of the algorithm running on $i = 1, 2, ...$ processors. We call S' the relative speed up, i.e. the speed up on p_1 processors is divided by the speed up on p_2 processors.

The relative speed up is equals to the usual speed up when $p_1 = 1$. Moreover the ideal relative speed up is

$$S'_{ideal} = \frac{p_2}{p_1}$$

The relative speed up can be used for measuring the performance gain when the reference value is not the execution time of the sequential algorithm ($p_2 > 1$). Now, if we compute the relative speed up using $p_2 = 24$ and $p_1 = 8$ we get:

$$\frac{S(24)}{S(8)} = 2.256$$

i.e. the parallel algorithm on $24 = 3 \cdot 8$ cores is scaling quite linearly from first ($p_1 = 8$) to second ($p_2 = 3 \cdot 8$) level of parallelism.
Figure 6 shows the execution time of OceanVAR software compared to that of the parallel version. We note that in this case by looking at the reduction of the total execution time we get a performance gain of about 60%, and this is in agreement with the improvement of the relative speedup S'.

6 Conclusions

This work describes computation efforts towards the development of a parallel software implementing the 3D-VAR/4D-VAR data assimilation. We start from analyzing the OceanVAR software, in terms of its condition number and describe the benefits obtained from using the Cholesky factorization, observing that the Truncated SVD provides better results. Moreover, we introduce the co-design of OceanVAR software, identifying the routines that have the greatest computational cost and introducing a prototype form of parallelism on two levels. In addition, including calls to scientific libraries for performing the most expensive computations enabled us to improve the accuracy and to make faster the code migration on different computing platforms. The performance

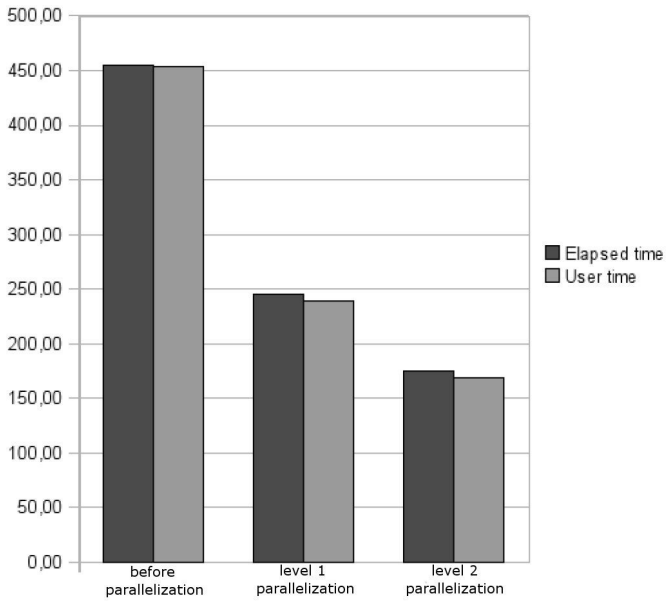

Figure 6: Execution time of OceanVar

gain obtained was shown by experimental results performed on IBM Power6 supercomputer, located in the laboratories of the CMCC - Ecotekne (Lecce).

This work follows the methodology that starts from the analysis of the available software, identifies the bottlenecks and introduces their solution on a real code employed in a production context. Of course it is a first step to develop a fully parallel software, and already the results have shown us the benefits that we can get. Those benefits suggest us to keep follow this path trying to get better results.
Software challenges are the employments of hierarchical algorithms to deal with bandwidth across the memory hierarchy, strategies to mitigate high memory latencies and the need for automated fault tolerance, performance analysis and verification.

References

[1] Blas - Blacs User Manual - Netlib *http://www.netlib.org/blas - blacs.*

[2] P. D'Ambra, D'Amore L. , A. Murli , *Parallel Computation and Problem Solving Methodologies: A view from some Experiences*, in Recent Trends in Numerical Analysis, in Advances in Computation: Theory and Practice, Nova Science Publisher, 2000. pp. 249-268.

[3] L. D'Amore, R. Arcucci, L.Marcellino and A. Murli, *A Parallel Three-dimensional Variational Data Assimilation Scheme*, Numerical Analysis and Applied Mathematics ICNAAM 2011 - AIP Conf. Proc. 1389, 1829-1831 (2011) American Institute of Physics.

[4] S.Dobricic, N.Pinardi, *An oceanographic three-dimensional variational data assimilation scheme*, 2008 Elsevier.

[5] S. Dobricic, N. Pinardi, M. Adani, A. Bonazzi, C. Fratianni and M. Tonani, *Mediterranean Forecasting System: An improved assimilation scheme for sea-level anomaly and its validation*, 16 January 2006

[6] R.M. Gray, *Toeplitz and Circulant matrices: A Review, Foundations and Trendsr on Communications and Information Theory*, 2, no.3, 2006, pp.155 239.

[7] Groetsch C.W., *The theory of Tikhonov regularization for Fredholm equations of the first kind*, Pitman Advanced Publishing Program

[8] GLOBAL OCEAN DATA ASSIMILATION: PROSPECTS & STRATEGIES, USGODAE Workshop, April 23-25, 2001 University of Maryland.

[9] S. A. Haben, A. S. Lawless and N. K. Nichols, *Conditioning of the 3DVAR Data Assimilation Problem*, Department of Mathematics University of Reading, September 2009

[10] C.Hansen, *Rank-Deficient and Discrete Ill-Posed Problems, numerical aspects of linear inversion*, SIAM, 1998

[11] E.Kalnay, *Atmospheric modeling, data assimilation and predictability*, 2003 Cambridge

[12] E.Kalnay, C.M.Danforth, *Using Singular Value Decomposition to Parameterize State-Dependent Model Errors*, May 11, 2007

[13] Lapack - Linpack User Manual - Netlib *http://www.netlib.org/lapack - linpack.*

[14] E.Lorenz *Empirical Orthogonal Functions and Statistical Weather Prediction*, december 1956, scientific report No.1, Statistical Forecasting Project.

[15] D. McLaughlin, A. O'Neill, J. Derber, M. Kamachi, Opportunities for enhanced collaboration within the data assimilation community, Q. J. R. Meteorol. Soc., 2005, 131, pp. 3683-3693

[16] A. Murli et al., *Some Perspective on High-Performance Mathematical Software*, in "High Performance Algorithms and Software in Nonlinear Optimization", Kluwer Academic Publishers, pp. 1-23, 1998.

[17] J. M. Navon, *Data Assimilation for Numerical Weather Prediction: A Review*, in "Data Assimilation for Atmospheric, Oceanic and Hydrologyc Applications. Springer-Verlag Berlin, 2009.

[18] N.K.Nichols, S.A.Haben, A.S.Lawless, *Conditioning and preconditioning of the variational data assimilation problem*, preprint MPS_2010_16, school of mathematics and metereology and physics, University of Reading.

[19] N. K. Nichols, *Matematical Concepts of Data Assimilation*, W. Lahoz et al eds. *Data Assimilation*, Springer-Verlag (Berlin), 2010.

[20] J. Nocedal R.H. Byrd, P. Lu and C. Zhu, *L-BFGS-B: Fortran Subroutines for Large-Scale Bound-Constrained Optimization*, ACM Transactions on Mathematical Software, Vol. 23, No. 4, December 1997, Pages 550-560.

[21] A. Rodrigues, S. Dosanjh, S. Hemmert, *Co-design for High Performance Computing*, AIP Conf. Proc. 1281, 1309 (2010).

[22] E. Agullo, B. Hadri, H. Ltaief and J. Dongarra, *Comparative Study of One-Sided Factorizations with Multiple Software Packages on Multi-Core Hardware*, LAPACKWorking Note #217.