Parallel Dynamic Domain Decomposition in Space - Time for Data Assimilation problems

L D’Amore* and R Cacciapuoti
University of Naples, “Federico II”, Complesso Universitario M. S. Angelo, via Cintia, 80126 Naples, Italy
*Email: luisa.damore@unina.it, rosalba.cacciapuoti@unina.it

Abstract. In the present work we employ a load balancing scheme involving an adaptive and dynamic workload redistribution both along Space and Time directions for solving Data Assimilation problems where the observations are non-uniformly distributed, general sparse and its distribution changes during the time. We will consider the Constrained Least Square model (CLS) as prototype of Data Assimilation problems and we will validate the proposed approach on different configurations. Validation is performed using Parallel Computing Toolbox of MATLABR2013a on high performance hybrid computing architecture.

1. Introduction
Data Assimilation (DA, for short) is the uncertainty quantification technique where observations/measurements of physical quantities are combined with model predictions to improve their estimates by minimizing inherent errors. This poses a formidable computational challenge and makes DA an example of big data inverse problems [1-4]. In this regard, in the following, we start considering Constrained Least Square (CLS) model as a prototype of DA problems [5]. In [6,7] we presented a feasibility analysis of a Domain Decomposition (DD) framework for solving time dependent CLS models in large scale application. In the present work we employ a load balancing scheme involving the adaptive and dynamic workload redistribution both along space and time directions. Namely, we address DA problems where the observations are non uniformly distributed, general sparse and its distribution changes during the time window. The present work is organized as follows. In section §2 we present DyDDST (Dynamic Domain Decomposition in Space and Time) through a schematic description and the numerical algorithm. Validation results are presented in Section §3 and, finally, in Section §4 we give conclusions.

2. DyDDST: The Dynamic DD in Space and Time framework
In many problems within the earth and environmental sciences observations are non uniformly distributed and its distribution change during time. DYDDST algorithm is proposed to support real time application where load measurement is necessary to determine when load imbalance occurs. We apply DyDDST on algorithm proposed in [8] in order to ensure a balanced distribution of load between spatial sub domains in each time interval. The load balancing scheme proposed involves, at each time
interval, an adaptive and dynamic repartitioning of load among spatial sub domains. Load repartition is performed by shifting boundaries of adjacent sub domains defined by the initial domain partitioning. As shown in Algorithm 1, DyDDST framework consists in five steps:

1) DD in space and time: $\Omega \times \Delta$ decomposition:
   i. Decomposition of $\Omega$ into a sequence of sub domains $\Omega_i$ such that $n_{sub} \in N$ is the number of sub domains.
   $$\Omega = \bigcup_{i = 1}^{n_{sub}} \Omega_i$$
   ii. Decomposition of time window $\Delta$, where $N_t \in N$ is the number of intervals.
   $$\Delta = \bigcup_{k = 1}^{N_t} \Delta_k$$

2) DD check: starting from the initial DD of $\Omega \times \Delta$, DYDDST performs a check of the partitioning. If a spatial sub domain is empty, it decomposes it in two sub domains the adjacent sub domain which has the maximum load.

3) Scheduling step: DyDDST computes the amount of observations needed for achieving, in each sub domain $\Omega_i$, the average load in $\Delta_k$; this is performed by using the connected graph $G^k$ associated to the DD of $\Omega$ in $\Delta_k$; i.e. $G^k$ depends on the configuration in $\Delta_k$ of spatial sub domains. This is achieved computing the Laplace matrix $L^k = \{L^k_{ij}\}$ as follows [8]:

$$L^k_{ij} = \begin{cases} 
-1 & i \neq j \text{ and } (i,j) \in G^k \\
\bar{d}^k(i) & i = j, \\
0 & \text{otherwise}
\end{cases}$$

(1)

and the load imbalance $b^k = (l^k(i) - \bar{l}^k)$, where $d^k(i)$ is the degree of vertex $i$ in $\Delta_k$; finally, $l^k(i)$ and $\bar{l}^k$ are the number of observations and the average load in $\Omega_i \times \Delta_k$, respectively. Solution of the Laplacian system:

$$L^k \lambda^k = b^k$$

(2)

associated to $G^k$ gives the amount of data which should have migrated in $\Delta_k$.

4) Migration step: DyDDST shifts the boundaries of spatial sub domains.

5) Updating step: DyDDST redefines spatial sub domains in $\Delta_k$ such that each one contains the number of observations computed in the scheduling step and redistributes sub domains among processors grids. After, for all sub domains it is necessary to re-evaluate the workload to balance the number of observations in $\Delta_{k+1}$.

3. Validation Results

Validation of DyDDST algorithm is performed on the high performance hybrid computing architecture of the SCoPE (Sistema Cooperativo Per Elaborazioni scientifiche multidisciplinari) data center, located in the University of Naples Federico II. Specifically, the HPC architecture is composed by 8 nodes which consist of distributed memory DELL M600 blades. The blades are connected by a 10 Gigabit Ethernet technology and each of them is composed of 2 Intel Xeon@2.33GHz quadcore processors sharing the same local 16 GB RAM memory for a number of 8 cores per blade and of 64 total cores. This is an intra-node configuration implementing a coarse-grained parallelization strategy on multiprocessor systems with
many-core CPUs. We study the performance of DyDDST algorithm by using Parallel Computing Toolbox of MATLABR2013a.

DyDDST set up. We define:
- \( \Omega \subset \mathbb{R}^2 \): spatial domain;
- \( N = 2048 \): mesh size;
- \( n_t = 64 \): number of elements of \( \Delta \);
- \( n = N \times n_t \): problem size;
- \( p \): number of spatial sub domains and processing units;
- \( N_t = p \): number of time intervals;
- \( i \): identification number of processing unit, which is the same of the associated sub domain;
- for \( k = 1, \ldots, N_t, m_k \): number of observations in \( \Delta_k \);
- \( \mathbf{m} = [m_1, \ldots, m_{N_t}] \in \mathbb{N}^{N_t} \): vector of number of observations in \( \Delta \);

iii. \( d^k(i) \): degree of \( i \) in \( \Delta_k \), i.e. number of sub domains adjacent to \( \Omega_i \) in \( \Delta_k \);
iv. \( l^k_{in}(i) \in N \): number of observations in \( \Omega_i \) in \( \Delta_k \) before the dynamic load balancing;
v. \( l^k_{out}(i) \in N \): number of observations in \( \Omega_i \) in \( \Delta_k \) after the dynamic load balancing.

- \( \text{Iter} = \max_{k=1, \ldots, N_t} (\text{Iter}(k)) \): maximum between the number of iterations needed to solve Laplacian system in (2) associated to \( G^k \) with Preconditioned Conjugate Gradient (PCG) method in each time interval \( \Delta_k \);
- \( T^p_{\text{DyDDST}}(\mathbf{m}) \): time (in seconds) needed to perform DyDDST on \( p \) processing units;
- \( T^p_r(\mathbf{m}) \): time (in seconds) needed to perform re-partitioning of \( \Omega \);
- \( O^p_{\text{DyDDST}}(\mathbf{m}) = \frac{T_r(\mathbf{m})}{T^p_{\text{DyDDST}}(\mathbf{m})} \): overhead time to the dynamic load balancing.

Regarding DD-DA, we let:
- \( \frac{n_{loc}}{p} \times \frac{n_t}{p} \): be local problem size;
- \( T^1(\mathbf{m}, n) \): sequential time (in seconds) needed to DA;
- \( T^p_{\text{DD-DA}}(\mathbf{m}, n_{loc}) \): time (in seconds) needed to perform in parallel DD-DA solving CLS problem after DyDDST procedure;
- \( S^p(\mathbf{m}, n_{loc}) = \frac{T^1(\mathbf{m}, n)}{T^p_{\text{DD-DA}}(\mathbf{m}, n_{loc})} \): algorithm speed-up;
- \( E^p(\mathbf{m}, n_{loc}) = \frac{S^p(\mathbf{m}, n_{loc})}{p} \): algorithm efficiency.

As measure of the load balance of DyDDST algorithm, for \( k = 1, \ldots, N_t \) we use [9]:
\[
\varepsilon^k = \frac{\min_i \left( l^k_{in}(i) \right)}{\max_i \left( l^k_{out}(i) \right)}
\]
i.e. we compute the ratio of the minimum to the maximum of the number of observations of sub domains \( \Omega_1, \ldots, \Omega_p \) in \( \Delta_k \) after applying DyDDST algorithm, respectively. Further, \( \varepsilon^k = 1 \) indicates a perfectly balanced system in \( \Delta_k \).

In the following tables we report results obtained by employing three configurations. More precisely, fixed \( p=4 \), for \( k=1, \ldots, 4 \) configuration considered in Example 1 changes in \( \Delta_k \) i.e. some sub domains are such
that its number of adjacent sub domains changes in $\Delta_k$, and the degree $d^k(i)$ of the vertex $i$ of processor graph changes. In Examples 2 and 3, for $p=2,4,8,32,64$ and $k=1,...,64$ configurations are the same in $\Delta_k$ while it changes the number of sub domains adjacent to each sub domain. Namely, in Example 2, sub domains $\Omega_1$ and $\Omega_p$ have 1 adjacent sub domain and for $i=2,...,p-1$, $\Omega_i$ has 2 adjacent sub domains while in Example 3 $\Omega_1$ has $p-1$ adjacent sub domains and for $i=2,...,p$, $\Omega_i$ has 1 adjacent sub domain.

From these experiments, we observe that the number of sub domains adjacent to each sub domain increases parameter $\xi^k$. On the contrary, as the number of adjacent sub domains increases, communications required by the workload repartitioning among sub domains increases, accordingly, the number of operations needed to compute the amount of observations required to obtain load balance increases, increasing $T_{DYDDST}(m)$. We are currently working for improving DYDDST algorithm so that it is able to effectively deal with the adaptive and dynamic load repartitioning along the entire time window.

**Example 1** (Tables 1-2) First configuration: $p=4$ spatial sub domains and time intervals such that:

1) $k=1$:
   - for $i=1,...,p$: $\Omega_i \times \Delta_k$: have data i.e. observations;
   - $d^k(1) = d^k(4) = 1, d^k(2) = d^k(3) = 2$.

2) $k=2$:
   - $\Omega_1 \times \Delta_k$: is empty;
   - for $j=2,3,4$: $\Omega_j \times \Delta_k$: have data;
   - $d^k(1) = d^k(2) = 2, d^k(3) = 1, d^k(4) = 3$.

3) $k=3$:
   - for $i=1,2$: $\Omega_i \times \Delta_k$: is empty;
   - for $j=3,4$: $\Omega_j \times \Delta_k$: have data;
   - $d^k(1) = d^k(4) = 1, d^k(2) = d^k(3) = 2$.

4) $k=4$:
   - for $i=1,2,3$: $\Omega_i \times \Delta_k$: is empty;
   - $\Omega_4 \times \Delta_k$: have data;
   - $d^k(1) = d^k(4) = 1, d^k(2) = d^k(3) = 2$.

| $p$ | $k$ | $m_k$ | $T_{DYDDST}(m_k)$ | $T_r(m_k)$ | $Oh_{PYDDST}(m_k)$ | $\xi^k$ | $\text{Iter}(k)$ |
|-----|-----|-------|-------------------|---------|----------------|------|-----------|
| 4   | 1   | 2217  | $2.58 \times 10^{-1}$ | 0       | $9.98 \times 10^{-1}$ | 9.98 | 1         |
|     | 2   | 2933  | $8.11 \times 10^{-2}$ | $8.00 \times 10^{-4}$ | $9.99 \times 10^{-4}$ | $9.99 \times 10^{-1}$ | 2       |
|     | 3   | 1925  | $7.05 \times 10^{-2}$ | $8.00 \times 10^{-4}$ | $1.13 \times 10^{-2}$ | $9.98 \times 10^{-1}$ | 2       |
|     | 4   | 1678  | $5.82 \times 10^{-2}$ | $1.20 \times 10^{-4}$ | $1.37 \times 10^{-2}$ | $9.98 \times 10^{-1}$ | 2       |

Table 1. Example 1. For $k=1$ all sub domains have data, consequently, it is not necessary to perform re-partitioning of $\Omega$ and $T_r(m_k) = 0$. 
We note that in Example 2 the number of sub domains which are adjacent to window, while in system in (2) (see Tables 3-4) are less than in Example 3.

Example 2 (Table 3 and Figure 1): We consider $p = 2, 4, 8, 16, 32, 64$ such that:

- $d^p(i) = d^p(1) = 1$: $\Omega_1$ and $\Omega_p$ have 1 adjacent sub domain in $\Delta_k$;
- for $i = 2, ..., p-1$, $d^p(i) = 2$: $\Omega_i$ has 2 adjacent sub domains in $\Delta_k$;
- $m_k$: number of observations available in $\Delta_k$ as defined in Table 5.

Example 3 (Table 4 and Figure 2) Second configuration: we consider $p = 2, 4, 8, 16, 32, 64$ such that: for $k = 1, ..., 64$

- $d^p(i) = p - 1$: $\Omega_4$ has $p - 1$ adjacent sub domains in $\Delta_k$;
- for $i = 2, ..., p$, $d^p(i) = 1$: $\Omega_i$ has 1 adjacent sub domain in $\Delta_k$;
- $m_k$: number of observations available in $\Delta_k$ defined in Table 5.

We note that in Example 2 the number of sub domains which are adjacent to $\Omega_4$ increases along the time window, while in Example 3 it equals to 2. Consequently, in Example 2, iterations needed to solve linear system in (2) (see Tables 3-4) are less than in Example 3.

Table 2. Example 1: We report values obtained by applying DD-DA after DyDDST in Example 1 (Table 1).

| $p$ | $n = 131072$ | $m = [1617 \ 2894 \ 1098 \ 2445]$ | $T^1(m, n) = 11.92 \times 10^9$ |
|-----|--------------|----------------------------------|--------------------------------|
| $p$ | $n_{loc}$ | $T^p_{DD-DA}(m, n_{loc})$ | $S^p(m, n_{loc})$ | $E^p(m, n_{loc})$ |
| 4   | 8192        | $3.45 \times 10^9$             | $3.46 \times 10^9$ | $8.65 \times 10^9$ |

Table 3. Example 2. Performance results.

| $p$ | $m$ in Table 5 | $T^1(m, n) = 6.41 \times 10^2$ |
|-----|----------------|--------------------------------|
| $p$ | $n_{loc}$ | $T^p_{DD-DA}(m)$ | $T^p_{DD-DA}(m, n_{loc})$ | $S^p(m, n_{loc})$ | $E^p(m, n_{loc})$ | Iter |
| 2   | 32768        | $9.34 \times 10^4$         | $3.62 \times 10^2$ | $1.78 \times 10^9$ | $8.89 \times 10^4$ | 1  |
| 4   | 8192         | $3.13 \times 10^0$         | $1.65 \times 10^2$ | $3.88 \times 10^8$ | $9.69 \times 10^4$ | 3  |
| 8   | 2048         | $8.12 \times 10^0$         | $9.95 \times 10^1$ | $6.65 \times 10^8$ | $8.07 \times 10^4$ | 7  |
| 16  | 512          | $1.65 \times 10^0$         | $5.70 \times 10^1$ | $1.11 \times 10^1$ | $6.93 \times 10^4$ | 15 |
| 32  | 128          | $3.08 \times 10^1$         | $3.26 \times 10^1$ | $1.98 \times 10^1$ | $6.15 \times 10^1$ | 20 |
| 64  | 32           | $5.77 \times 10^1$         | $3.18 \times 10^1$ | $2.02 \times 10^1$ | $3.56 \times 10^1$ | 20 |

Table 4. Example 3. Performance results.

| $p$ | $n = 131072$ | $m$ in Table 5 | $T^1(m, n) = 6.41 \times 10^2$ |
|-----|--------------|----------------|--------------------------------|
| $p$ | $n_{loc}$ | $T^p_{DD-DA}(m)$ | $T^p_{DD-DA}(m, n_{loc})$ | $S^p(m, n_{loc})$ | $E^p(m, n_{loc})$ | Iter |
| 2   | 32768        | $9.34 \times 10^4$         | $3.62 \times 10^2$ | $1.78 \times 10^9$ | $8.89 \times 10^4$ | 1  |
| 4   | 8192         | $2.44 \times 10^6$         | $1.65 \times 10^2$ | $3.90 \times 10^8$ | $9.74 \times 10^4$ | 2  |
| 8   | 2048         | $6.72 \times 10^6$         | $1.02 \times 10^2$ | $6.31 \times 10^8$ | $7.89 \times 10^4$ | 2  |
| 16  | 512          | $1.21 \times 10^6$         | $5.53 \times 10^1$ | $1.16 \times 10^1$ | $7.26 \times 10^4$ | 2  |
Table 5. Example 2-3. For \( k = 1, \ldots, 64 \), values of \( m_k \) in \( \Delta_k \).

| \( k \) | \( m_k \) | \( k \) | \( m_k \) | \( k \) | \( m_k \) | \( k \) | \( m_k \) | \( k \) | \( m_k \) | \( k \) | \( m_k \) |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1618 | 9 | 2327 | 17 | 1579 | 25 | 2651 | 33 | 1209 | 41 | 2256 |
| 2 | 2419 | 10 | 1678 | 18 | 2744 | 26 | 2571 | 34 | 2626 | 42 | 1343 |
| 3 | 2523 | 11 | 1739 | 19 | 1493 | 27 | 2174 | 35 | 1683 | 43 | 2048 |
| 4 | 2869 | 12 | 1968 | 20 | 1912 | 28 | 2555 | 36 | 2332 | 44 | 2667 |
| 5 | 2260 | 13 | 2078 | 21 | 2890 | 29 | 1603 | 37 | 1691 | 45 | 2687 |
| 6 | 1874 | 14 | 2512 | 22 | 2439 | 30 | 963 | 38 | 2146 | 46 | 2184 |
| 7 | 2036 | 15 | 2613 | 23 | 2214 | 31 | 2270 | 39 | 233 | 47 | 2763 |
| 8 | 2536 | 16 | 2584 | 24 | 2476 | 32 | 2611 | 40 | 1772 | 48 | 2392 |

Figure 1. Example 2. For \( k = 1, \ldots, 64 \), value of parameter \( \mathcal{E}^k \) in \( \Delta_k \).
4. Conclusions
An essential role in the success of domain decomposition approaches is to maintain a nearly equal number of data among sub domains. We employed a load balancing scheme based on the adaptive and dynamic redistribution of load among spatial sub domains in each time interval. We presented first results obtained by applying DyDDST to time dependent CLS problems using different configurations of the initial DD. From the experiments, we note that DYDDST efficiency strongly depends on the degree of the vertices of processors graph in each time interval i.e. on the number of sub domains adjacent to each sub domain in each time interval. According to [10], we are improving DYDDST algorithm so that it is able to effectively deal with the adaptive and dynamic load repartitioning in different time intervals.
Algorithm 1. Procedure DyDDST.

Procedure DyDDST-Dynamic Load Balancing in Space and Time (in: $p, N, \Omega, \Delta$ out: $l_1, ..., l_p$)
% Procedure DyDDST allows to balance observations between adjacent sub domains in $\Delta$. Domain $\Omega \times \Delta$ is % decomposed in $p \times N$ sub domains and some of spatial sub domains may be empty.

Initial DD step
% DD of $\Omega \times \Delta$ in $(\Omega_1, \Omega_2, ..., \Omega_p)$ and $(\Delta_1, \Delta_2, ..., \Delta_N)$
end of Initial DD step

DD step
Define $n_i$, the number of sub domains adjacent to $\Omega_i$
Define $l_i^k$: the amount of observations in $\Omega_i \times \Delta_k$
repeat
% identification of $\Omega_m$, the adjacent sub domain to $\Omega_i$ with the maximum load
Compute $l_i^k = \max_{j=1,...,n_i} (l_j)$: the maximum amount of observations
Decompose $\Omega_m$ in 2 sub domains: $\Omega_m \rightarrow (\Omega_1, \Omega_2)$
end of DD Step

Begin Scheduling step
Define $G^k$: the graph associated with initial partition of $\Omega \times \Delta_k$: vertex $i$ corresponds to $\Omega_i$ in $\Delta_k$
Distribute the amount of observations $l_i^k$ in $\Omega_i$
Define $d^k(i) = n_i$, the degree of node $i$ of $G^k$
repeat
Compute the average load: $\bar{l}_i^k = \frac{\sum_{j=1}^{n_i} l_j^k}{n_i}$
Compute load imbalance: $b_i^k = (l_i^k - \bar{l}_i^k)_{i=1,...,p}$
Compute $L^k, \text{Laplacian matrix of } G^k$
Call PCG(in: $L^k, b^k$, out: $x^k$) % Preconditioned Conjugate Gradient algorithm solving the linear system $L^k x^k = b^k$
Compute $\delta^k_{ij}$, the load increment between two adjacent sub domains.
Define $n_{i_1}^k, n_{i_2}^k$, number of those sub domains whose configuration has to be updated
Update $G^k$
Update amount of observations in $\Omega_i$: $l_i^k = l_i^k - \sum_{j=1}^{n_i} \delta^k_{ij} + \sum_{j=1}^{n_i} \delta^k_{ji}$
until (max $|l_i^k - \bar{l}_i^k| = \frac{\text{deg}(i)}{2}$) i.e. maximum load-difference is $d^k(i)/2$
end Scheduling step

Begin Migration Step
Shift boundaries of two spatial adjacent sub domains in order to achieve a balanced load in $\Delta_k$.
end Migration Step

Update step
Update DD
Update $l_i^k \equiv l_i^{k+1}$ the number of observations of sub domain $\Omega_i$ in $\Delta_{k+1}$ (not yet balanced)
end Update step
Define $l_i \equiv l_i^k$ on $\Omega_i \times \Delta_k$.

end Procedure DyDDST
References

[1] Antonelli L, Carracciulo L, Ceccarelli M, D’Amore L, Murli A 2002 Total Variation Regularization for Edge Preserving 3D SPECT Imaging in High Performance Computing Environments. In: Sloot P.M.A., Hoekstra A G, Tan C J K, Dongarra J J (eds) Computational Science — ICCS 2002. Lecture Notes in Computer Science, 2330, (Springer, Berlin, Heidelberg)

[2] Arcucci R, D’Amore L, Celestino S, Laccetti G, Murli A 2016 A Scalable Numerical Algorithm for Solving Tikhonov Regularization Problems. In: Parallel Processing and Applied Mathematics, Lecture Notes in Computer Science, 9574, (Heidelberg, Springer) pp 45-54, ISBN: 978-3-319-32152-3, ISSN: 0302-9743, doi: 10.1007/978-3-319-32152-3-5.

[3] Arcucci R, D’Amore L, Carracciulo L 2015 On the problem-decomposition of scalable 4D-Var Data Assimilation model, Proceedings of the 2015 International Conference on High Performance Computing and Simulation, HPCS, pp 589-594

[4] Arcucci R, D’Amore L, Carracciulo L, Scotti G, Laccetti G 2017 A Decomposition of the Tikhonov Regularization Functional oriented to exploit hybrid multilevel parallelism, International Journal of Parallel Programming, 45 1214-1235

[5] Gander W 1980 Least squares with a quadratic constraint Numerische Mathematik 36 291-307

[6] D’Amore L, Cacciapuoti R and V. Mele 2019 Ab initio Domain Decomposition Approaches for Large Scale Kalman Filter Methods: A case study to Constrained Least Square Problems, 13th International Conference, PPAM 2019, (Bialystok, Poland) September 8-11, 2019, 10.1007/978-3-030-43222-5, LNCS, 12044

[7] D’Amore L, Cacciapuoti R, Mele V. 2020 A scalable Kalman filter algorithm. Trustworthy analysis on constrained least square model, Concurrency: Practice and Experience, e6022 doi: https://doi.org/10.1002/cpe.6022.

[8] Hu Y F, Blake R J and Emerson D R 1998 - An optimal migration algorithm for dynamic load balancing, Concurrency: Practice and Experience, 10(6) 467-483

[9] Kohring G A 1998 Dynamic load balancing for parallelized particle simulations on MIMD computers Parallel Computing 21 683-693

[10] Murli A, Boccia V, Carracciulo L, D’Amore L, Laccetti G, Lapegna M 2007 Monitoring and Migration of a PETSc-based Parallel Application for Medical Imaging in a Grid Computing PSE IFIP International Federation for Information Processing, 239, Grid-Based Problem Solving Environments, Springer, 421–432, ISBN 978-0-387-73658-7.