



# Fast Data-Obtaining Algorithm for Data Assimilation with Large Data Set

Junmin Xiao<sup>1,2</sup>  · Guizhao Zhang<sup>3</sup> · Yanan Gao<sup>4</sup> · Xuehai Hong<sup>3</sup> · Guangming Tan<sup>1,2</sup>

Received: 4 August 2019 / Accepted: 28 November 2019 / Published online: 6 December 2019  
© Springer Science+Business Media, LLC, part of Springer Nature 2019

## Abstract

Data assimilation is an analysis technique which combines observations and the numerical results from theoretical models to deduce more realistic and accurate data. It is widely used in investigations of the atmosphere, ocean and land surface. Due to the complicated data structure of the inputs from dynamical models and the increase of the amount of model data, the parallelization of data assimilation suffers from high overhead on file reading and data communication. In this paper, we propose a flexible parallel data access approach for reading a large number of data from disks firstly. Using this approach, the data access conflict is avoided successfully, and the frequency of disk addressing operations is also decreased significantly. Next, we design a communication-avoiding strategy to reduce the communication volume at the cost of some additional computations. Furthermore, we present a “pipe-flow” scheme for data exchange to conduct conflict-free message passing. Consequently, a fast data-obtaining algorithm is developed for the data assimilation. Our experiments show that the fast data-obtaining algorithm gains a performance of  $5\times$  speedup compared with the baseline, which is excellent at data-obtaining for the parallel data assimilation. Due to the reduction of disk addressing operations, the new approach achieves  $6\times$  speedup on average for the file reading process. Since a large amount of data movement can be avoided, the new approach achieves  $2.7\times$  speedup on average for the communication between processors.

**Keywords** Data assimilation · I/O optimization · Communication optimization · Parallel implementation · Domain localization

---

The work is supported by the National Key Research and Development Program of China under Grant No. 2016YFC1401706 and National Natural Science Foundation of China under Grant No. 61802369.

✉ Junmin Xiao  
xiaojunmin@ict.ac.cn

Extended author information available on the last page of the article

## 1 Introduction

Data assimilation is a powerful technique which has been widely applied in investigations of the atmosphere [17,20], ocean [11,30], and land surface [24]. It combines observation data and the underlying dynamical principles governing the system to provide an estimate of the state of the system which is better than the observation or the model result alone [10]. In recent years, with the development of high resolution data assimilation, the size of each sample becomes very large, and the number of observation data is increasing significantly [6]. Consequently, real applications usually need to deal with the data with hundreds or thousands of GB. Hence, the data assimilation becomes a scientific big data problem.

In order to execute the data assimilation parallelly in a system with several processors, the domain localization strategy is commonly used [16,29], which updates the value of each physical variable on a point of 2D or 3D space by only considering observations within a given radius of influence  $r$  [15]. Based on this basic idea, the processors could assimilate data on different subdomains simultaneously. The computation time would reduce almost linearly with the number of processors increasing. Consequently, the time for computation can be ignored, while the time for reading data would dominate the main part of total runtime. Since the high resolution data analysis requires a large number of input data, the assimilation process can easily be bottlenecked by an inefficient approach to bring input data to processors. In the recent decay, many I/O optimization methods have been developed to speed up the process of file reading in the data assimilation [2], such as the concurrent access approach, the in-memory workflow method and so on. They give good solutions for different situations respectively, and are widely used in applications.

In this work, we consider a special data assimilation which is widely used in real applications to analyze the simulation results from dynamical models, such as Hybrid Coordinate Ocean Model (HYCOM). In this assimilation process, not all simulation results from dynamical models are involved. The useful input data is discontinuously stored in different locations of disks. It would result in a large number of disk addressing operations and the long time for file reading. On the other hand, in the data assimilation for HYCOM, the data analysis of each physical variable on a mesh point only depends on few observations, which leads to the computation time is much less than the I/O time. Therefore, it is difficult to improve the performance by overlapping I/O and computation similar to recent work [27].

In order to efficiently optimize the data-obtaining process in the data assimilation for real applications, this work would do a deeper investigation of the computation workflow. Inspired by the concurrent access approach [27], we also choose some processors only for file reading, and divide the data-obtaining process into two stages: file reading and data communication. Furthermore, the corresponding optimization approaches for I/O and data movement are considered respectively based on the in-depth analysis of bottlenecks brought by the application requirements.

In this work, we make the following key contributions:

- Propose a flexible parallel data access approach for reading a large number of data from disks. Using this approach, the data access conflict is avoided successfully, and the frequency of disk addressing operations is also decreased significantly.
- Design a communication-avoiding strategy to reduce the communication volume at the cost of some additional computations.
- Present a “pipe-flow” scheme for data exchange to conduct conflict-free message passing.
- Develop a fast data-obtaining algorithm for data assimilation, based on our optimization strategies for file reading and data communication.
- Evaluate the performance of the fast data-obtaining algorithm with large data sets, which demonstrates the high scalability and significant performance improvement on the data-obtaining process.

## 2 Background and Related Work

### 2.1 Hybrid Coordinate Ocean Model

The Hybrid Coordinate Ocean Model (HYCOM) [8,28] was developed from the Miami Isopycnic Coordinate Ocean Model [3,4,7]. It is characterized by a hybrid vertical coordinate that transfers smoothly from the isopycnal coordinate in the open, stratified ocean to the terrain-following sigma coordinate in the coastal regions and to the  $z$  coordinate in the mixed layer and unstratified seas. Such setup may reasonably simulate coastal or open-sea ocean states by combining the advantages of different types of coordinates. The K-profile parameterization vertical mixing scheme [18] is included in HYCOM. The model domain spans the Pacific and Indian oceans from  $27^\circ$  E to  $290^\circ$  E and from  $50^\circ$  S to  $60^\circ$  N with several vertical hybrid layers. The HYCOM is forced by the 6-hourly fields, which include temperature, dew point temperature, mean sea level pressure, and wind. The lateral boundary conditions and sea surface salinity fields are relaxed toward monthly climatologies taken from the Generalized Digital Environmental Model [25]. In this work, HYCOM is one of data sources. The useful inputs for data assimilation are a part of direct results from HYCOM, and they are discontinuously stored in disks.

### 2.2 Assimilation Method

Data assimilation is an analysis technique in which the observed information is accumulated into the model state by taking advantage of consistency constraints with laws of time evolution and physical properties. Recent decades have witnessed the development of the assimilation technique [5,13]. Several assimilation methods have been proposed, such as the optimal interpolation, the Kalman filter [14,22], variational methods (3D-Var [1,19] and 4D-Var [9,23]), and so on. The assimilation method considered in this paper, is Ensemble Optimal Interpolation (EnOI) method [12,21].

A simulation result provided the dynamical model such as HYCOM usually contains  $M$  physical characteristics ( $M \gg 1$ ), while there are just  $m$  characteristics which

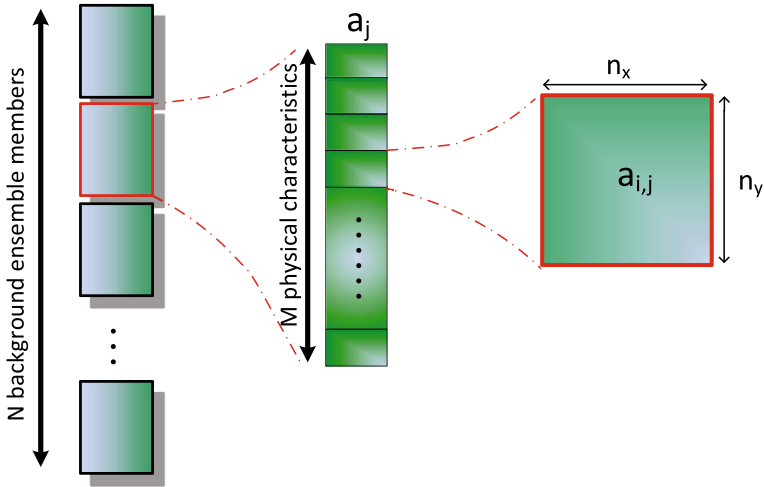


Fig. 1 Data structure of background ensemble members

would be needed for the data assimilation ( $m < M$ ). Without loss of generality, assume that the useful characteristics are the first  $m$  ones. In our algorithm design, we will discuss how to obtain  $m$  out of  $M$  characteristics in general cases. After getting the  $m$  characteristics, the first  $m_1$  ones are used to correct the latter  $m_2$  characteristics ( $m = m_1 + m_2$ ). On the  $n_x \times n_y$  longitude–latitude mesh of the earth surface,  $\varphi_i \in \mathbb{R}^n$  and  $\psi_i \in \mathbb{R}^n$  represent the analysis field and the background field for the  $i$ th physical characteristic respectively, and  $n = n_x \cdot n_y$  is the size of the model state vector. Let  $d_i \in \mathbb{R}^{h_i}$  be the perturbed observation for the  $i$ th physical characteristic, and  $h_i$  is the number of measurements. Based on the statistical error correction, EnOI method gives the solution as follows

$$\varphi_i = \psi_i + F(A_i, \psi_1, \dots, \psi_{m_1}, d_1, \dots, d_{m_1}) \in \mathbb{R}^n, \quad i = m_1 + 1, \dots, m_1 + m_2, \tag{1}$$

where the function  $F(\cdot)$  is the analysis increment, and  $A_i$  is the background ensemble for  $i$ th physical characteristic taken from the model integration.  $A_i$  can be written as

$$A_i = [a_{i,1}, a_{i,2}, \dots, a_{i,N}] \in \mathbb{R}^{n \times N}.$$

Let  $a_j = [a_{1,j}, a_{2,j}, \dots, a_{M,j}]$ .  $a_j$  is called the  $j$ th background ensemble member ( $j = 1, 2, \dots, N$ ).  $N$  is the number of ensemble members. For each  $a_j$ , the element  $a_{i,j} \in \mathbb{R}^n$  can be represented as

$$a_{i,j} = (a_{i,j}[1], a_{i,j}[2], \dots, a_{i,j}[n])^T,$$

in which  $a_{i,j}[k]$  is the  $k$ th element value of  $a_{i,j}$  on the  $k$ th mesh point of the longitude–latitude mesh. Figure 1 shows the data structure of background ensemble members. Furthermore, the background error covariance matrix can be estimated by

$$B_i = \tilde{A}_i \tilde{A}_i^T, \quad (2)$$

where

$$\tilde{A}_i = [\tilde{a}_{i,1}, \tilde{a}_{i,2}, \dots, \tilde{a}_{i,N}] = \frac{1}{\sqrt{N-1}} (A_i - \bar{a}_i \otimes \mathbf{1}_N^T). \quad (3)$$

Here,  $\bar{a}_i$  is the ensemble mean,  $\mathbf{1}_N^T \in \mathbb{R}^N$  is a vector whose  $N$  components are all ones, and  $\otimes$  denotes the outer product of two vectors. The superscript  $T$  denotes matrix transpose. By using the first  $m_1$  background fields  $\psi_i$  ( $i = 1, 2, \dots, m_1$ ), a measurement error can be presented as

$$U = \begin{pmatrix} d_1 - H(\psi_1, d_1) \\ d_2 - H(\psi_2, d_2) \\ \vdots \\ d_{m_1} - H(\psi_{m_1}, d_{m_1}) \end{pmatrix}, \quad (4)$$

and  $H$  is an observation operator which maps from the model space to the observation space. In general, the analysis increment of EnOI reads

$$F(A_i, \psi_1, \dots, \psi_{m_1}, d_1, \dots, d_{m_1}) = \alpha \tilde{A}_i S^T (\alpha S S^T + R)^{-1} U, \quad (5)$$

where

$$S = (S_1, S_2, \dots, S_N), \text{ and } S_j = \begin{pmatrix} H(\tilde{a}_{1,j}, d_1) \\ H(\tilde{a}_{2,j}, d_2) \\ \vdots \\ H(\tilde{a}_{m_1,j}, d_{m_1}) \end{pmatrix}. \quad (6)$$

In the Eq. (5),  $R$  is the estimated data error covariance matrix, and the parameter  $\alpha$  is a scalar for tuning the magnitude of the covariance.

### 2.3 Parallel Implementation

In order to execute the data assimilation parallelly in a system with several processors, the localization is commonly used [16,29], which updates the value of each physical variable on a mesh point by only considering observations within a given radius of influence  $r$  such as 400 km. Based on this basic idea, the domain decomposition is proposed, in which the whole computation domain is split into  $n_{sd}$  subdomains. Assume that there are also  $n_{sd}$  computation processors in the system, and each processor is responsible for the local data update in a subdomain. For the  $k$ th subdomain  $D^k$ , we define the expansion  $\bar{D}^k$  as the point set which includes the subdomain  $D^k$  and all additional points needed for local computation. For instance, if the  $k$ th subdomain is  $D^k = [x_1^k, x_2^k] \times [y_1^k, y_2^k]$ , then the expansion of  $D^k$  can be represented as  $\bar{D}^k = [x_1^k - \xi, x_2^k + \xi] \times [y_1^k - \eta, y_2^k + \eta]$ . Denote  $\psi_i^k$  and  $\bar{\psi}_i^k$  as the constraints of  $\psi_i$

on  $D^k$  and  $\bar{D}^k$  respectively. For example,  $\psi_i^k = (\psi_i^k[j_1], \dots, \psi_i^k[j_{h_k}]) \in \mathbb{R}^{h_k}$  where  $h_k$  is the number of mesh points in  $D^k$ , and all  $j_1$ th,  $\dots$ ,  $j_{h_k}$ th mesh points are in  $D^k$ . Similarly, we can define  $d_i^k, a_{i,j}^k$  on  $D^k$ , and  $\bar{d}_i^k, \bar{a}_{i,j}^k$  on  $\bar{D}^k$  respectively. According to the Eqs. (1) and (5), the analysis field on  $D^k$  can be represented as

$$\phi_i^k = \psi_i^k + F(A_i^k, \psi_1^k, \dots, \psi_{m_1}^k, d_1^k, \dots, d_{m_1}^k), \quad i = m_1 + 1, \dots, m_1 + m_2, \tag{7}$$

where

$$F(A_i^k, \psi_1^k, \dots, \psi_{m_1}^k, d_1^k, \dots, d_{m_1}^k) = \alpha \tilde{A}_i^k (S^k)^T (\alpha (S^k)(S^k)^T + R^k)^{-1} (U^k).$$

here  $R^k$  is the estimated data error covariance matrix on the  $k$ th subdomain.  $\tilde{A}_i^k$  can be calculated by

$$\tilde{A}_i^k = [\tilde{a}_{i,1}^k, \tilde{a}_{i,2}^k, \dots, \tilde{a}_{i,N}^k], \quad \text{where } \tilde{a}_{i,j}^k = \frac{a_{i,j}^k - \bar{a}_i}{\sqrt{N-1}}. \tag{8}$$

Moreover,

$$S^k = (S_1^k, S_2^k, \dots, S_N^k), \quad \text{where } S_j^k = \begin{pmatrix} H(\bar{a}_{1,j}^k, \bar{d}_1^k) \\ H(\bar{a}_{2,j}^k, \bar{d}_2^k) \\ \vdots \\ H(\bar{a}_{m_1,j}^k, \bar{d}_{m_1}^k) \end{pmatrix}, \tag{9}$$

and

$$U^k = \begin{pmatrix} d_1^k - H(\bar{\psi}_1^k, \bar{d}_1^k) \\ d_2^k - H(\bar{\psi}_2^k, \bar{d}_2^k) \\ \vdots \\ d_{m_1}^k - H(\bar{\psi}_{m_1}^k, \bar{d}_{m_1}^k) \end{pmatrix}. \tag{10}$$

In each subdomain, after the necessary data is acquired, the local analysis (7) is computed independently, and then all results are mapped back onto the global domain.

The parallel implementation of the data assimilation is shown in Alogrithm 1. First of all, the  $k$ th processor has to obtain the observations  $\bar{d}_i^k$ , the characteristics  $\bar{a}_{i,j}^k$ , and the background field  $\bar{\psi}_i^k$  ( $i = 1, 2, \dots, m_1, j = 1, 2, \dots, N$ ) on the  $k$ th expansion  $\bar{D}^k$ . Secondly,  $S^k$  and  $U^k$  are calculated according to the Eqs. (9) and (10) respectively. Using  $\bar{a}_{i,j}^k$ ,  $R^k$  is constructed. Thirdly, the  $k$ th processor reads other background ensemble elements  $\bar{a}_{i,j}^k$  on  $\bar{D}^k$  and background fields  $\psi_i^k$  on  $D^k$  with  $i = m_1 + 1, \dots, m_1 + m_2$  and  $j = 1, 2, \dots, N$ . Finally,  $\tilde{A}_i^k$  follows from the Eq. (8). Furthermore, the Eq. (7) leads to the final result  $\phi_i^k$ .

**Algorithm 1** Parallel Data Assimilation

**Require:** Background ensemble members  $a_j$  ( $j = 1, 2, \dots, N$ ), and background fields  $\psi_i$ , perturbed observations  $d_i$  ( $i = 1, 2, \dots, M$ ), and observation operator  $H$ ;

**Ensure:** Analysis field  $\varphi_i^k$ ;

1: // The  $k$ -th processor obtains data on the expansion of  $k$ -th subdomain ( $k = 1, 2, \dots, n_{sd}$ );

2: Obtain  $\bar{a}_{i,j}^k$  from  $a_j$  with  $i = 1, 2, \dots, m_1$ ,  $j = 1, 2, \dots, N$ ;

3: Obtain  $\bar{\psi}_i^k$  from  $\psi_i$  with  $i = 1, 2, \dots, m_1$ ;

4: Obtain  $\bar{d}_i^k$  from  $d_i$  with  $i = 1, 2, \dots, m_1$ ;

5: //The  $k$ -th processor executes local computation;

6: Calculate  $H(\bar{a}_{i,j}^k, \bar{d}_i^k)$ ;

7: Deduce  $S^k$  according to (9);

8: Deduce  $U^k$  according to (10);

9: Construct  $R^k$  using  $\bar{a}_{i,j}^k$ ;

10: //The  $k$ -th processor obtains data on the expansion of  $k$ -th subdomain;

11: Obtain  $\bar{a}_{i,j}^k$  from  $a_j$  with  $i = m_1 + 1, \dots, m_1 + m_2$ ,  $j = 1, 2, \dots, N$ ;

12: Obtain  $\bar{\psi}_i^k$  from  $\psi_i$  with  $i = m_1 + 1, \dots, m_1 + m_2$ ;

13: //The  $k$ -th processor executes local computation;

14: **for**  $i = m_1 + 1$  to  $m_1 + m_2$  **do**

15:   Deduce  $\bar{A}_i^k$  according to (8);

16:    $\varphi_i^k = \bar{\psi}_i^k + \alpha \bar{A}_i^k (S^k)^T (\alpha (S^k)(S^k)^T + R^k)^{-1} (U^k)$ ;

17:   Write  $\varphi_i^k$  into disks;

18: **end for**

19: Return;

**2.4 Parallel Optimization**

Two main parts in Algorithm 1 are local computation and data-obtaining. The local computation involves the singular value decomposition to realize the matrix inverse, which takes up most of the total runtime as the number of processors is small. However, the time for local computation can decrease almost linearly with the number of processors increasing, which leads to the decrease of the size (or number of mesh points) of each subdomain. On the other hand, the background ensembles  $A_i$  with about hundreds of GB have to be read from disks. If  $A_i^k$  are read directly from disks by multiple processors in parallel, the data access conflict would occur. When the number of processors increases, I/O time would continue to rise. Therefore, the file reading process is the performance bottleneck of the data assimilation.

In recent works, the in-memory workflow method is commonly used for I/O-intensive tasks. In this way, all processors are divided into two parts: I/O processors and computation processors. All data are read by I/O processors and stored in their memory. The computation processors would fetch data from I/O processors. If the data have to be used repeatedly, the in-memory workflow only needs to read the data from disks once. All data always flow within the memory of different processors, which could avoid frequent I/O operations. Although the in-memory workflow method is widely applied in some I/O libraries such as PIO, it is not helpful for improving I/O behaviour of Algorithm 1. The reason is that, in Algorithm 1, all data are used only once, and the I/O time is much longer than the computation time.

In order to avoid multiple processors accessing a disk at the same time, the concurrent access approach has been proposed [27]. This approach also uses some I/O processors for file reading, which seems similar to the in-memory workflow. But, the motivation of the concurrent access approach is to transform the task of the data assimilation from I/O-intensive one to CPU-intensive, which provides an opportunity for overlapping I/O and computation. However, in the application of Algorithm 1 for HYCOM, the data analysis of each physical variable on a mesh point only depends on few observations. Consequently, the time for local computation can be ignored compared with the I/O time. Hence, it is impossible to use the concurrent access approach to change Algorithm 1 from I/O-intensive to CPU-intensive. Even though the optimization strategy may be designed for overlapping I/O and computation, the overlap ratio must be small. In this work, we would focus on the design to improve the data-obtaining process in Algorithm 1.

### 3 Algorithm Design

In this section, we describe our algorithm design for the parallel data assimilation. Firstly, we propose a flexible parallel data access approach for reading the background ensembles on each subdomain. Next, we design a communication-avoiding strategy to reduce the communication volume at the cost of some additional computations. Thirdly, we present a “pipe-flow” scheme for data exchange to conduct conflict-free message passing. Finally, our algorithm design leads to a new algorithm, which is called as Fast Data-obtaining Algorithm.

#### 3.1 Parallel Data Access Approach

There are two main challenges in the designing of the file reading approach. On one hand, in the recent file system, a disk only supports a limited number of processors to access it simultaneously. If a lot of processors would access the same disk at the same time, many processors have to wait for the disk resource to become available. Hence, it is inefficient to use all processors for reading data from disks directly. Consequently, the first challenge is how to avoid the data access conflict. On the other hand, in the real applications, each ensemble member obtained from the dynamical model, usually has  $M$  physical characteristics, while the data assimilation only involves  $m$  of them. This fact means that, for the  $j$ th background ensemble member  $a_j = [a_{1,j}, a_{2,j}, \dots, a_{M,j}]$ , the useful  $m$  elements have to be selected from the set  $\{a_{1,j}, a_{2,j}, \dots, a_{M,j}\}$ . In the file system, each ensemble  $a_j$  is stored as an independent file. If the whole ensemble  $a_j$  are read into the memory, it is likely to bring large I/O volume and cause memory overflow. If the processors only read the useful  $m$  elements of  $a_j$ , the file reading process would involve a discontinuous data access, which results in many disk addressing operations and an inefficient disk access. Therefore, the second challenge is how to decrease the amount of disk addressing operations when the useful  $m$  elements of  $a_j$  are read.

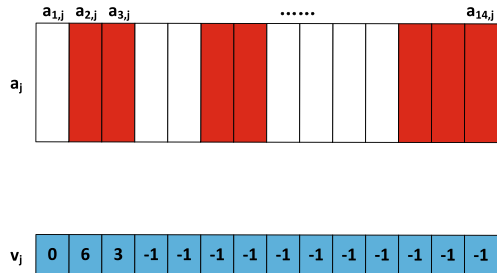
In order to solve two bottlenecks above for file reading in the data assimilation, we propose a parallel data access approach as follows. Firstly, inspired by the concurrent



```

input: double t1, double t2, double * a_j;
Output: int * v_j;
01: for ( int k = 1; k <= M; k ++ ){
02:     v_j[k]=-1;
03: }
04: int nStart = -1;
05: int h = 0;
06: int k = 1;
07: if ( a_{1,j} is useless ) {
08:     v_j[1]=0;
09:     k = 2;
10: }
11: for ( int i = 1; i <= M; i ++ ){
12:     if ( a_{i,j} is useless ){
13:         h += 1;
14:     }
15:     else{
16:         h = 0;
17:     }
18:     if ( h == 0 && nStart == -1 ){
19:         nStart = i - 1;
20:     }
21:     if ( h * t1 > t2 && nStart != -1 ){
22:         v_j[k] = i - nStart - h;
23:         k += 1;
24:         nStart = -1;
25:         h = 0;
26:     }
27: }

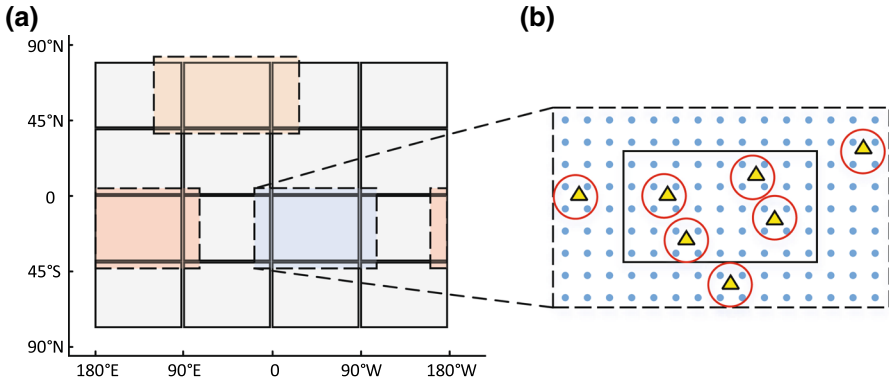
```



**Fig. 2** Code skeleton for determining each element of  $v_j$  and an example with  $m = 7$  and  $M = 14$ . For a background ensemble  $a_j$ , the red and white blocks represent the useful and needless physical characteristics respectively (Color figure online)

access approach in [27], we also choose  $q$  processors ( $q \geq 1$ ) only for file reading (I/O processors) and the other processors for local analysis (Computation processors). Each I/O processor only needs to access  $N/q$  ensembles. Secondly, the I/O processor runs a test program to obtain the approximate values of time  $t_1$  for reading a vector with length of  $n$  such as  $a_{i,j}$ , and time  $t_2$  for one disk addressing. Based on  $t_1$ ,  $t_2$  and the input information which contains the indexes of the useful  $m$  elements in  $a_j$ , the I/O processor for accessing  $a_j$  would construct a vector  $v_j$  which determines how the processor access  $a_j$  (Fig. 2). If an element of  $v_j$  equals to “ $K$ ” ( $K \geq 0$ ), the I/O processor needs to read the concurrent continuous  $K$  elements of  $a_j$ , such as  $a_{i_0+1,j}, a_{i_0+2,j}, \dots, a_{i_0+K,j}$ , and then starts a new disk addressing to find the next useful data. If an element of  $v_j$  is equal to “ $-1$ ”, the I/O processor stops the data access on  $a_j$ . Thirdly, after getting the necessary data  $a_{i,j}$ , the I/O processor splits it into several small ones  $\bar{a}_{i,j}^k$ , and sends  $\bar{a}_{i,j}^k$  to the computation processor which is responsible for the data assimilation on the  $k$ th subdomain  $D^k$ .

In the approach above, the I/O processors should be chosen according to the computational environment. In a distributed-memory system, a node usually has several cores. In order to take full advantage of the memory and network bandwidth of nodes, it is efficient to use one core of a node as the I/O processor. Since the number of I/O processors can be fixed, the new approach effectively avoids a lot of processors accessing a disk during the total number of processors becomes large. Furthermore, although the new approach also has to read some needless data, it could effectively decrease the I/O time and the frequency of disk addressing.



**Fig. 3** Localization. **a** the domain decomposition. **b** the  $k$ th subdomain and its expansion. The solid and dashed boxes represent the  $k$ th subdomain  $D^k$  and the expansion  $\bar{D}^k$  respectively. The yellow triangles show the observation points. Red circles highlight the positions where the data of  $\bar{a}_{i,j}^k$  on blue mesh points and  $\bar{d}_i^k$  on yellow observations points are necessary for calculating  $H(\bar{a}_{i,j}^k, \bar{d}_i^k)$  (Color figure online)

### 3.2 Communication-Avoiding Strategy

To calculate  $H(\bar{a}_{i,j}^k, \bar{d}_i^k)$  (or  $H(\bar{\psi}_i^k, \bar{d}_i^k)$ ), the computation processors have to read  $\bar{a}_{i,j}^k$  (or  $\bar{\psi}_i^k$ ) and  $\bar{d}_i^k$  from disks directly, or receive them from I/O processors. On one hand, since the size of each whole perturbed observation  $d_i$  is small, the computation processors can obtain  $d_i$  fast from disks in the initial stage when the I/O processors access  $a_{i,j}$ . After getting  $d_i$ , the  $k$ th computation processor only needs to screen out  $\bar{d}_i^k$  from  $d_i$ . On the other hand, for each  $D^k = [x_1^k, x_2^k] \times [y_1^k, y_2^k]$ , its expansion is  $\bar{D}^k = [x_1^k - \xi, x_2^k + \xi] \times [y_1^k - \eta, y_2^k + \eta]$ . Since the values of  $\xi$  and  $\eta$  are fixed, the halo area  $\bar{D}^k / D^k$  would dominate most part of the expansion  $\bar{D}^k$  during the number of subdomains increases and the size of  $D^k$  decreases (Fig. 3). This fact would result in the significant increment of communication volume when the number of processors enlarges.

In order to control the communication volume, we do a deep investigation of the calculation of  $H(\bar{a}_{i,j}^k, \bar{d}_i^k)$ . We find that  $H(\bar{a}_{i,j}^k, \bar{d}_i^k)$  only involves the  $\bar{a}_{i,j}^k$  around observation points (the red circles highlight the areas in Fig. 3b). This fact implies that the volume of the necessary data for the computation of  $H(\bar{a}_{i,j}^k, \bar{d}_i^k)$  is small. If I/O processors just send the necessary data, a large number of data movement can be avoided. Consequently, a communication-avoiding strategy is proposed as follows. Firstly, the computation processors read  $d_i$  from disks, and then screen out  $\bar{d}_i^k$  from  $d_i^k$ . Secondly, the  $k$ th computation processor calculates the positions of all observation points in the expansion  $\bar{D}^k$ . The  $l$ th observation point corresponds to the place where the  $l$ th element of  $\bar{d}_i^k$  is observed (the yellow triangles in Fig. 3b). Furthermore, the computation processors send the location information of observation points to I/O processors. Based on the locations, I/O processors choose the elements of  $\bar{a}_{i,j}^k$  on the mesh points around the observation points (in the red circles of Fig. 3b), and send them to the  $k$ th computation processor. Finally, the  $k$ th computation processor calculates  $H(\bar{a}_{i,j}^k, \bar{d}_i^k)$ .

By the way, when the number of processors is less than one thousand, it is reasonable for the conventional way to use the computation processor to receive the whole data of  $\bar{a}_{i,j}^k$  from I/O processors, because the computation of positions of observation points in  $\bar{D}^k$  is time-consuming. However, if thousands of processors are used, the size of each subdomain is not very large. Hence, the search process for the positions of observation points will be completed in seconds. Furthermore, using the location information of observation points, the I/O processors only need to send very little data to the computation processors for calculating  $H(\bar{a}_{i,j}^k, \bar{d}_i^k)$ , which achieves the communication-avoiding successfully.

### 3.3 “Pipe-Flow” Scheme for Data Exchange

After reading  $a_{i,j}$  from the disk, the I/O processor needs to split  $a_{i,j}$  into  $n_{sd}$  some small data  $\bar{a}_{i,j}^k$  ( $k = 1, 2, \dots, n_{sd}$ ), and send  $\bar{a}_{i,j}^k$  to the  $k$ th computation processor for the local computation on the  $k$ th subdomain, which requires a carefully designed communication strategy to conduct a conflict-free message passing. For example, we should avoid concurrent sending of data from several I/O processors to the same computation processor, otherwise both the number of message and the volume of data received by this processor are much larger other computation processors, which results in an imbalanced usage of the network.

Therefore, it is important to specify an optimized sequence of communications. For that purpose, we design a “pipe-flow” scheme as shown in Fig. 4. In the “pipe-flow” scheme, there are several different steps in the arrangement of communications. Communications are done like a flow from an I/O processor to different computation processors (Fig. 4). First of all, all computation processors are assigned to  $q$  groups. Next, In the  $k$ th step ( $1 \leq k \leq q$ ), the  $i$ th I/O processor broadcasts data to the computation processors in the  $j$ th group, where  $j = i + k - 1$  if  $i + k - 1 \leq q$ , otherwise  $j = i + k - 1 - q$ . After  $q$  steps, the  $k$ th computation processor would get all  $\bar{a}_{i,j}^k$  ( $j = 1, 2, \dots, N$ ) for the  $i$ th physical characteristic.

### 3.4 Fast Data-Obtaining Algorithm

Based on the discussion above, a fast data-obtaining algorithm for data assimilation is proposed (Algorithm 2), where some processors (I/O processors) are chosen for file reading, the others (Computation processors) are for the local updates on different subdomains.

Firstly, I/O processors read  $m_1$  useful background ensemble elements  $a_{i,j}$  from  $a_j = [a_{1,j}, a_{2,j}, \dots, a_{M,j}]$  in disks ( $i = i_1, i_2, \dots, i_{m_1}$ ). By using the parallel data access approach, a large number of disk addressing operations are avoided successfully. Furthermore,  $m_1$  background fields  $\psi_i$  also could be obtained from disks efficiently ( $i = i_1, i_2, \dots, i_{m_1}$ ). After obtaining  $a_{i,j}$  and  $\psi_i$ , I/O processors could screen out  $\bar{a}_{i,j}^k$  and  $\bar{\psi}_i^k$  on the expansion  $\bar{D}^k$  of the  $k$ th subdomain respectively. Moreover, I/O processors do not rush to send  $\bar{a}_{i,j}^k$  and  $\bar{\psi}_i^k$  to the  $k$ th computation processor, but just wait to receive the information about the positions of all observation points in  $\bar{D}^k$ .

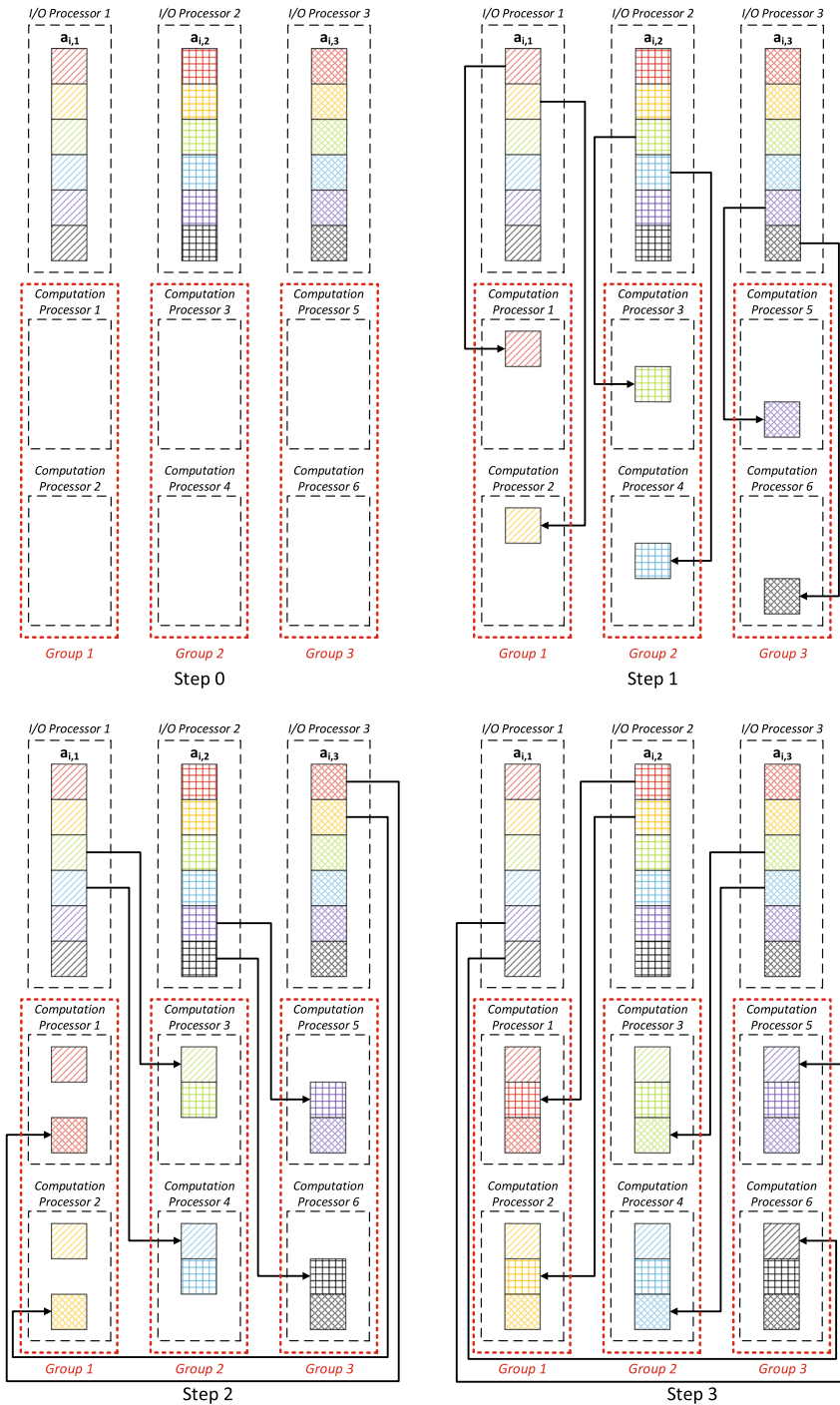


Fig. 4 "Pipe-flow" scheme for data exchange

---

**Algorithm 2** Fast Data-obtaining Algorithm
 

---

**Require:** Background ensemble members  $a_j$  ( $j = 1, 2, \dots, N$ ), and background fields  $\psi_i$ , perturbed observations  $d_i$  ( $i = 1, 2, \dots, M$ ), and observation operator  $H$ ;

**Ensure:** Analysis field  $\varphi_i^k$ ;

- 1: // I/O processors read data from disks;
- 2: Read the background elements  $a_{i,j}$  from  $a_j$  in disks using Parallel Data Access Approach ( $i = i_1, i_2, \dots, i_{m_1}$ ,  $j = 1, 2, \dots, N$ );
- 3: Read the background fields  $\psi_i$  from disks directly ( $i = i_1, i_2, \dots, i_{m_1}$ );
- 4: // I/O processors execute computation;
- 5: Screen out  $\tilde{a}_{i,j}^k$  from  $a_{i,j}$  ( $i = i_1, i_2, \dots, i_{m_1}$ ,  $j = 1, 2, \dots, N$ );
- 6: Screen out  $\tilde{\psi}_i^k$  from  $\psi_i$  ( $i = i_1, i_2, \dots, i_{m_1}$ );
- 7: // The  $k$ -th computation processor reads data from disks;
- 8: Read  $d_i$  from disks directly ( $i = i_1, i_2, \dots, i_{m_1}$ );
- 9: // The  $k$ -th computation processor executes local computation;
- 10: Screen out  $\tilde{d}_i^k$  from  $d_i$  ( $i = i_1, i_2, \dots, i_{m_1}$ );
- 11: Calculate the positions of all observation points in the expansion  $\tilde{D}^k$  for avoiding a large amount of communication volume;
- 12: // The  $k$ -th computation processor communicates with I/O processors;
- 13: Send the location information of observation points to I/O processors;
- 14: // I/O processors communicate with the  $k$ -th computation processor;
- 15: Receive the locations of the observation points in the expansion  $\tilde{D}^k$ ;
- 16: // I/O processors execute computation;
- 17: Choose the elements of  $\tilde{a}_{i,j}^k$  on the mesh points around the observation points;
- 18: // I/O processors communicate with computation processors;
- 19: Send the chosen elements of  $\tilde{a}_{i,j}^k$  to the  $k$ -th computation processor using “Pipe-flow” Scheme ( $i = i_1, i_2, \dots, i_{m_1}$ ,  $j = 1, 2, \dots, N$ );
- 20: Send  $\tilde{\psi}_i^k$  to the  $k$ -th computation processor using “Pipe-flow” Scheme ( $i = i_1, i_2, \dots, i_{m_1}$ );
- 21: // The  $k$ -th computation processor communicates with I/O processors;
- 22: Receive the necessary elements of  $\tilde{a}_{i,j}^k$  from I/O processors ( $i = i_1, i_2, \dots, i_{m_1}$ ,  $j = 1, 2, \dots, N$ );
- 23: Receive  $\tilde{\psi}_i^k$  from I/O processors ( $i = i_1, i_2, \dots, i_{m_1}$ );
- 24: // The  $k$ -th computation processor executes local computation;
- 25: Calculate  $H(\tilde{a}_{i,j}^k, \tilde{d}_i^k)$ ;
- 26: Deduce  $S^k$  according to (9);
- 27: Deduce  $U^k$  according to (10);
- 28: Construct  $R^k$  using  $\tilde{a}_{i,j}^k$ ;
- 29: // I/O processors read data from disks;
- 30: Read other background ensemble elements  $a_{i,j}$  from  $a_j$  in disks using Parallel Data Access Approach ( $i = i_{m_1+1}, \dots, i_{m_1+m_2}$ ,  $j = 1, 2, \dots, N$ );
- 31: Read other background fields  $\psi_i$  from disks directly ( $i = i_{m_1+1}, \dots, i_{m_1+m_2}$ );
- 32: // I/O processors execute computation;
- 33: Screen out  $\tilde{a}_{i,j}^k$  from  $a_{i,j}$  ( $i = i_{m_1+1}, \dots, i_{m_1+m_2}$ ,  $j = 1, 2, \dots, N$ );
- 34: Screen out  $\tilde{\psi}_i^k$  from  $\psi_i$  ( $i = i_{m_1+1}, \dots, i_{m_1+m_2}$ );
- 35: // I/O processors communicate with computation processors;
- 36: Send  $\tilde{a}_{i,j}^k$  to the  $k$ -th computation processor using “Pipe-flow” Scheme ( $i = i_{m_1+1}, \dots, i_{m_1+m_2}$ ,  $j = 1, 2, \dots, N$ );
- 37: Send  $\tilde{\psi}_i^k$  to the  $k$ -th computation processor using “Pipe-flow” Scheme ( $i = i_{m_1+1}, \dots, i_{m_1+m_2}$ );
- 38: // The  $k$ -th computation processor communicates with I/O processors;
- 39: Receive  $\tilde{a}_{i,j}^k$  from I/O computation processors ( $i = i_{m_1+1}, \dots, i_{m_1+m_2}$ ,  $j = 1, 2, \dots, N$ );
- 40: Receive  $\tilde{\psi}_i^k$  from I/O computation processors ( $i = i_{m_1+1}, \dots, i_{m_1+m_2}$ );
- 41: // The  $k$ -th computation processor executes local computation;
- 42: **for**  $h = m_1 + 1$  to  $m_1 + m_2$  **do**
- 43:    $i = i_h$ ;
- 44:   Deduce  $\tilde{A}_i^k$  according to (8);
- 45:    $\varphi_i^k = \psi_i^k + \alpha \tilde{A}_i^k (S^k)^T (\alpha (S^k)(S^k)^T + R^k)^{-1} (U^k)$ ;
- 46:   Write  $\varphi_i^k$  into disks;
- 47: **end for**
- 48: Return;

---

Secondly, when I/O processors screen out  $\bar{a}_{i,j}^k$  and  $\bar{\psi}_i^k$ , the  $k$ th computation processor can read  $m_1$  perturbed observations  $d_i$  from disks directly ( $i = i_1, i_2, \dots, i_{m_1}$ ), which is a fast procedure due to the small size of  $d_i$ . After getting  $d_i$ , the  $k$ th computation processor would screen out  $\bar{d}_i^k$  from  $d_i$ . Based on  $\bar{d}_i^k$ , it calculates the positions of all observation points in the  $k$ th expansion  $\bar{D}^k$ , and then sends the location information of observation points to I/O processors, which could make sure I/O processors to only provide the  $k$ th computation processor the necessary data. The motivation of our communication-avoiding design is to reduce the communication volume through some additional computations. Furthermore, since it is time-consuming to deduce the positions of all observation points, we use all computation processors to execute the tasks on different expansions in parallel, rather than using the limited number of I/O processors.

Thirdly, after receiving the locations of the observation points in the expansion  $\bar{D}^k$ , I/O processors could choose the elements of  $\bar{a}_{i,j}^k$  on the mesh points around the observation points, and then send the useful elements to the  $k$ th computation processor, which can reduce the communication volume significantly compared with communicating the whole  $\bar{a}_{i,j}^k$ . Meanwhile,  $\bar{\psi}_i^k$  are also sent. In order to avoid an imbalanced usage of the network and the conflict of message passing, the “pipe-flow” scheme is used for data exchange. As soon as the necessary elements of  $\bar{a}_{i,j}^k$  and  $\bar{\psi}_i^k$  are obtained, the  $k$ th computation processor can deduce  $S^k$ ,  $U^k$  and  $R^k$ .

Fourthly, I/O processors further read other  $m_2$  background ensemble elements  $a_{i,j}$  and  $m_2$  background fields  $\psi_i$  from disks using the parallel data access approach ( $i = i_{m_1+1}, \dots, i_{m_1+m_2}$ ). Next, they screen out  $\bar{a}_{i,j}^k$  and  $\psi_i^k$  from  $a_{i,j}$  and  $\psi_i$  respectively, and then send  $\bar{a}_{i,j}^k$  and  $\psi_i^k$  to the  $k$ th computation processor using the “pipe-flow” scheme.

Finally, after obtaining  $\bar{a}_{i,j}^k$  and  $\psi_i^k$  ( $i = i_{m_1+1}, \dots, i_{m_1+m_2}$ ), the  $k$ th computation processor executes the local analysis on the subdomain  $D^k$  and deduce  $\phi_i^k$ . This local data assimilation is done at each subdomain in parallel.

## 4 Evaluation

In this section, we present the performance evaluations of Fast Data-obtaining Algorithm (FDA) on Tianhe-2 supercomputer.

### 4.1 Computational Environment and Data Set

Tianhe-2 was the fastest supercomputer in the world from 2013 to 2015. It is equipped with two Intel Ivy Bridge CPUs (24 cores) for every node, and its interface nodes are connected using InfiniBand networks. The system software includes a 64-bit Kylin OS, an Intel 14.0 compiler, a customized MPICH-3.1 for TH Express-2, and a self-designed hybrid hierarchy file system H2FS. In our numerical experiments, we have used 204 nodes of Tianhe-2, where 24 nodes are only for I/O process, and the others are for local computation. Furthermore, MPI-IO library is installed in Tianhe-2 for parallel I/O processes. Hence, MPI-IO library is used in the file reading process of both FDA and our baseline.

In our test, the data assimilation involves 120 background ensemble members from HYCOM ( $N = 120$ ), and over 100,000 observations are used. Each ensemble member contains 314 physical characteristics ( $M = 314$ ). Not all 314 physical characteristics need to be considered in the data analysis. Only 117 of the characteristics are necessary in the assimilation process ( $m = 117$ ). Based on the statistical regularity from the first 30 characteristics ( $m_1 = 30$ ), the assimilation process adjusts the latter 87 characteristics ( $m_2 = 87$ ). Besides, the 120 background ensemble members are taken from a long-time ocean model integration with the  $0.1^\circ$  spatial resolution and 30 vertical levels. In the 2-dimensional latitude–longitude mesh,  $n_x \times n_y = 3600 \times 1800$ .

### 4.2 Baseline

In the performance evaluation, the baseline is chosen from [26], which is widely used for data assimilation. First of all, the baseline divides all processors into  $n_g$  groups. In each group, a processor (I/O agent) is selected for file reading. Secondly, each background ensemble member is divided into  $n_g$  overlapped bars along latitude direction. Each I/O agent is responsible for reading a bar respectively. Thirdly, I/O agent splits the data bar into several overlapped small blocks, and then broadcasts different blocks to different processors within its group. Finally, after obtaining all local data, each processor only needs to execute the local analysis on some subdomain respectively. It is worth noting that, this approach uses MPI-level communication to broadcast data, but the communication process is fast due to the size of each block being small.

In order to verify the high performance of the baseline, we compare the baseline with an original algorithm, in which all processors use MPI-IO library to read the necessary data directly and then do the local analysis respectively. As shown in Fig. 5, the baseline is faster than the original algorithm. With the number of processors increasing, the

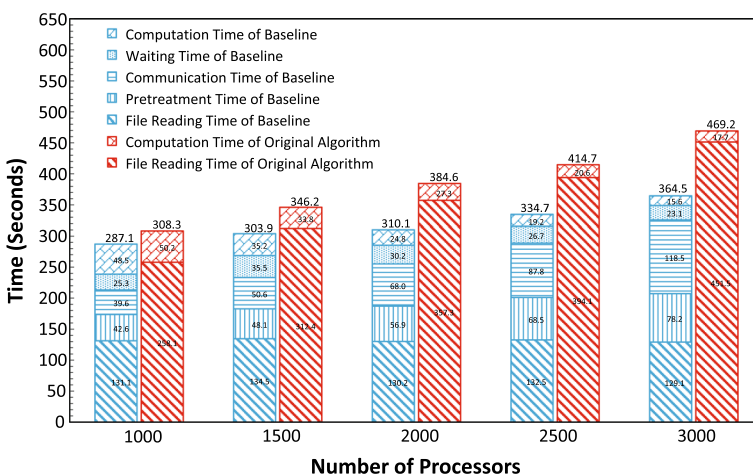


Fig. 5 Runtime of baseline and original algorithm. In the original algorithm, all processors directly use MPI-IO library for file reading

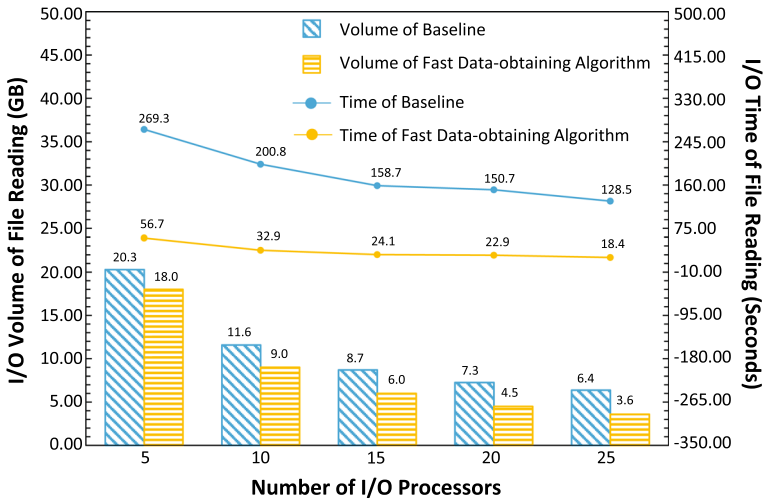


Fig. 6 Volume and time of file reading

performance gap between two algorithms becomes wider, which results from the uncontrollable increment of I/O time in the original algorithm. Even though MPI-IO library is used, the file reading process in the original algorithm has a poor behaviour. The main reason is that the data access conflict would become serious as a growing number of processors take part in the file reading process. However, in the baseline, the fixed number of processors are used for file reading. Therefore, its I/O time remains stable throughout, which indicates that the baseline is excellent at data-obtaining for the parallel data assimilation. This is our reason for choosing it as the baseline. It is clear that, during the total number of processors becomes larger, the time for both pretreatment and communication processes increases fast, which is one of performance deficiencies of the baseline. The improvements of the baseline would involve our contributions in this work.

### 4.3 Volume and time of file reading

In Fig. 6, the average I/O volume in FDA is about 85% of that in the baseline, while the average I/O time of FDA is less than 31 s, which is only 1/6 of the baseline's I/O time. The main reason is that, the time for disk addressing dominates the total I/O time, and our optimization decreases the frequency of disk addressing significantly. In the test, the number of disk addressing in the baseline is 14,040, while it equals to 20 in FDA. This fact indicates that the number of disk addressing is reduced by 99.9% in the new approach. Moreover, the I/O volume in FDA decreases proportionally with the number of I/O processors increasing, because each I/O processor in FDA can flexibly avoid the access to useless elements of  $a_j$ . However, the I/O volume decreases slowly in the baseline.



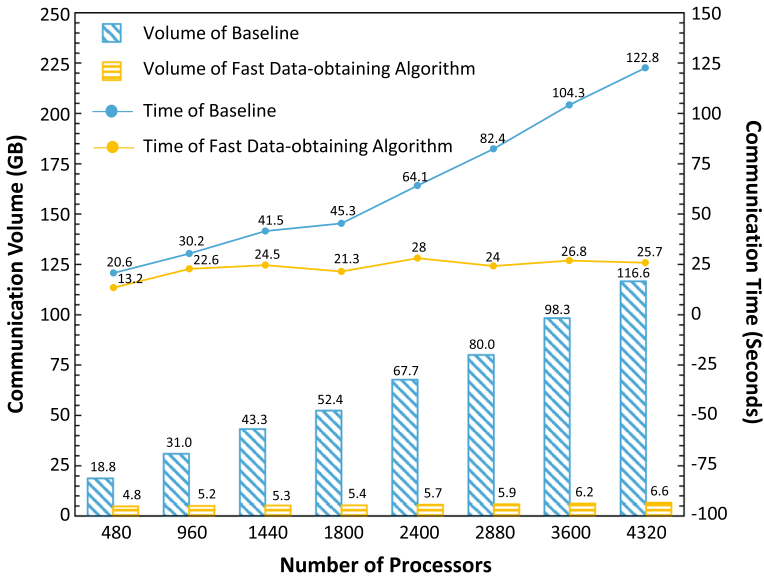


Fig. 7 Communication volume and time

#### 4.4 Communication Volume and Time

Figure 7 shows the communication volume and time varying with the number of processors increasing. When the number of processors equals to 4320, compared with the baseline, FDA reduces the total communication volume by 110 GB which is 94% of the communication volume of the baseline, and decreases the communication time by 97.1 s which is 80% of the communication time of the baseline. The main reason is that our communication-avoiding strategy avoids a large number of data movements successfully. On the other hand, when the number of processors exceeds 1800, both the communication volume and time of the baseline increase almost linearly. However, as the number of processors increases from 960 to 4320, both the communication volume and time of FDA change slightly with only small fluctuations, which demonstrates the good communication efficiency of FDA.

#### 4.5 Total Time for Data-Obtaining

The performance of different methods on the data-obtaining process is shown in Fig. 8. In this experiment, 24 I/O processors and 480 computation processors are used in each method. When the number of ensemble members increases from 20 (19 GB) to 120 (116 GB), the total time for file reading, communication and pretreatment in the baseline increases by 121 s, while it only goes up by 11 s in FDA. When the data volume reaches 116 GB, FDA saves 133 s compared with the baseline, which nearly equals to 80% of the data-obtaining time in the baseline. Furthermore, the trend in Fig. 8 shows that the performance gap between the two algorithms continues to widen

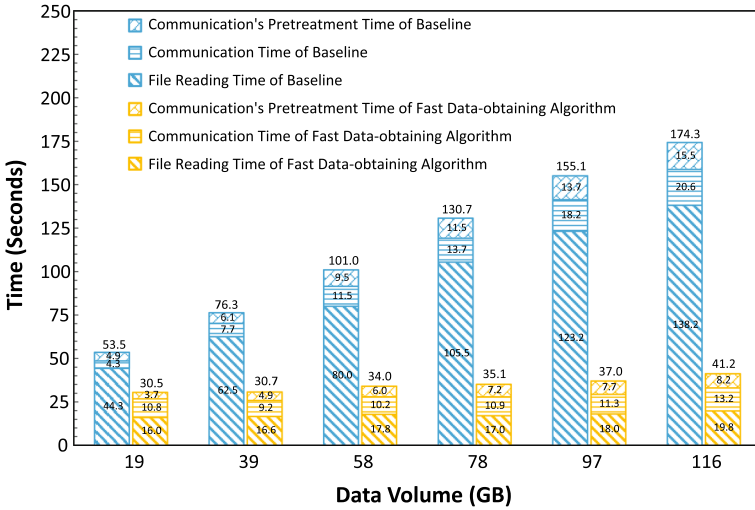


Fig. 8 Total time for data-obtaining

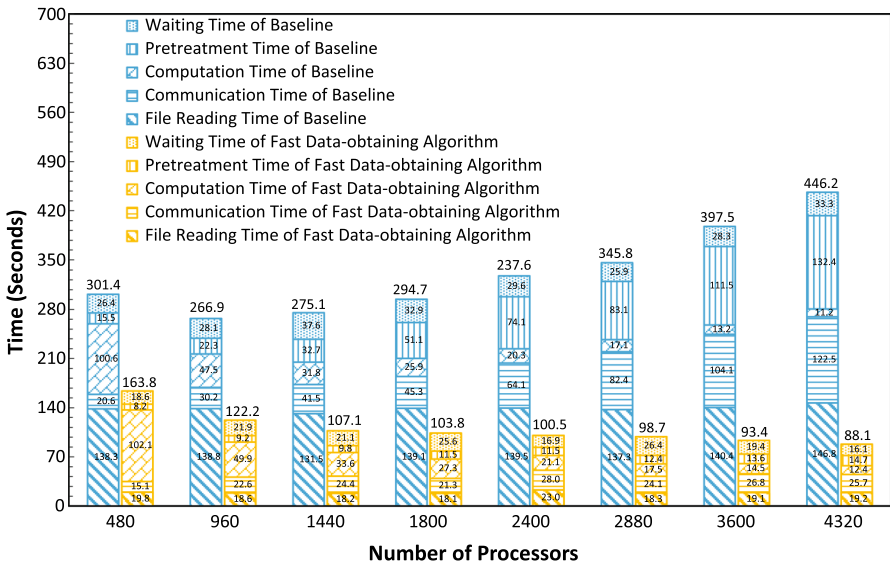


Fig. 9 Overall performance

as the data volume increases further, which demonstrates the good performance of the new approach on data-obtaining for large data sets.

### 4.6 Overall Performance

The overall performance is shown in Fig. 9. For the baseline, when the number of processors increases from 480 to 4320, the time for file reading is always more than

130 s. Meanwhile, both the communication time and the time of pretreatment for communication increase from about 20 to 130 s respectively. Although the computation time decreases with the number of processors increasing, the runtime of the baseline becomes longer when the number of processors is greater than 960. However, in FDA, the I/O process always takes less than 25 s. As the number of processors increases from 480 to 4320, the computation time reduces from 102 to 11 s, while the communication time and the time of pretreatment for communication increase by less than 10 s respectively. When 4320 processors are used, the total runtime of FDA is just 88.1 s, which achieves  $5\times$  speedup compared with the baseline.

## 5 Conclusion

In this paper, we have designed a fast data-obtaining algorithm for the data assimilation which is usually applied in real applications. By proposing the flexible parallel data access approach, a large number of disk addressing operations are avoided successfully. Through some additional computations for searching the positions of observation points, we design a communication-avoiding strategy to significantly reduce the communication volume. Furthermore, by considering a “pipe-flow” scheme for data exchange, we conduct conflict-free message passing. The experimental evaluation demonstrates the good scalability of the new algorithm and its significant performance improvement on the data-obtaining process.


**Acknowledgements** The authors would like to thank all anonymous reviewers for their valuable comments and helpful suggestions.

## References

1. Bei, N., de Foy, B., Lei, W., Zavala, M., Molina, L.T.: Using 3DVAR data assimilation system to improve ozone simulations in the Mexico city basin. *Atmos. Chem. Phys.* **8**(24), 7353–7366 (2008)
2. Bibov, A., Haario, H.: Parallel implementation of data assimilation. *Int. J. Numer. Methods Fluids* **83**(7), 606–622 (2017)
3. Bleck, R.: An oceanic general circulation model framed in hybrid isopycnic-Cartesian coordinates. *Ocean Model.* **4**(1), 55–88 (2002)
4. Bleck, R., Dean, S., O’Keefe, M., Sawdey, A.: A comparison of data-parallel and message-passing versions of the miami isopycnic coordinate ocean model (micom). *Parallel Comput.* **21**(10), 1695–1720 (1995)
5. Bloom, S.C., Takacs, L.L., Da Silva, A.M., Ledvina, D.: Data assimilation using incremental analysis updates. *Mon. Weather Rev.* **124**(6), 1256–1271 (1996)
6. Boniface, K., Ducrocq, V., Jaubert, G., Yan, X., Brousseau, P., Masson, F., Champollion, C., Chéry, J., Doerflinger, E.: Impact of high-resolution data assimilation of GPS Zenith delay on Mediterranean heavy rainfall forecasting. *Ann. Geophys.* **27**, 2739–2753 (2009)
7. Chassignet, E.P., Hurlburt, H.E., Smedstad, O.M., Halliwell, G.R., Hogan, P.J., Wallcraft, A.J., Baraille, R., Bleck, R.: The hycom (hybrid coordinate ocean model) data assimilative system. *J. Mar. Syst.* **65**(1–4), 60–83 (2007)
8. Chen, Y., Yan, C., Zhu, J.: Assimilation of sea surface temperature in a global hybrid coordinate ocean model. *Adv. Atmos. Sci.* **35**(10), 1291–1304 (2018)
9. Clayton, A.M., Lorenc, A.C., Barker, D.M.: Operational implementation of a hybrid ensemble/4d-var global data assimilation system at the met office. *Quart. J. R. Meteorol. Soc.* **139**(675), 1445–1461 (2013)

10. Dee, D.P., Uppala, S.M., Simmons, A.J., Berrisford, P., Poli, P., Kobayashi, S., Andrae, U., Balsameda, M.A., Balsamo, G., Bauer, P., et al.: The era-interim reanalysis: configuration and performance of the data assimilation system. *Quart. J. Roy. Meteorol. Soc.* **137**(656), 553–597 (2011)
11. Emerick, A.A., Reynolds, A.C.: Ensemble smoother with multiple data assimilation. *Comput. Geosci.* **55**, 3–15 (2013)
12. Evensen, G.: The ensemble Kalman filter: theoretical formulation and practical implementation. *Ocean Dyn.* **53**(4), 343–367 (2003)
13. Ghil, M., Malanotte-Rizzoli, P.: Data assimilation in meteorology and oceanography. In: Dmowska, R., Saltzman, B. (eds.) *Advances in Geophysics*, vol. 33, pp. 141–266. Elsevier, Amsterdam (1991)
14. Houtekamer, P.L., Mitchell, H.L.: A sequential ensemble Kalman filter for atmospheric data assimilation. *Mon. Weather Rev.* **129**(1), 123–137 (2001)
15. Hunt, B.R., Kostelich, E.J., Szunyogh, I.: Efficient data assimilation for spatiotemporal chaos: a local ensemble transform Kalman filter. *Phys. D* **230**(1), 112–126 (2007)
16. Keppenne, C.L.: Data assimilation into a primitive-equation model with a parallel ensemble Kalman filter. *Mon. Weather Rev.* **128**(6), 1971–1981 (2000)
17. Kucharski, F., Molteni, F., Bracco, A.: Decadal interactions between the western tropical pacific and the north atlantic oscillation. *Clim. Dyn.* **26**(1), 79–91 (2006)
18. Large, W.G., McWilliams, J.C., Doney, S.C.: Oceanic vertical mixing: a review and a model with a nonlocal boundary layer parameterization. *Rev. Geophys.* **32**(4), 363–403 (1994)
19. Li, Y., Wang, X., Xue, M.: Assimilation of radar radial velocity data with the WRF hybrid ensemble-3DVAR system for the prediction of Hurricane ike. *Mon. Weather Rev.* **140**(11), 3507–3524 (2012)
20. Molteni, F.: Atmospheric simulations using a GCM with simplified physical parametrizations. I: model climatology and variability in multi-decadal experiments. *Clim. Dyn.* **20**(2), 175–191 (2003)
21. Oke, P.R., Brassington, G.B., Griffin, D.A., Schiller, A.: The bluelink ocean data assimilation system (bodas). *Ocean Model.* **21**(1–2), 46–70 (2008)
22. Ott, E., Hunt, B.R., Szunyogh, I., Zimin, A.V., Kostelich, E.J., Corazza, M., Kalnay, E., Patil, D.J., Yorke, J.A.: A local ensemble Kalman filter for atmospheric data assimilation. *Tellus A Dyn. Meteorol. Oceanogr.* **56**(5), 415–428 (2004)
23. Powell, B.S., Arango, H.G., Moore, A.M., Lorenzo, E.D., Milliff, R.F., Foley, D.: 4DVAR data assimilation in the intra-americas sea with the regional ocean modeling system (ROMS). *Ocean Model.* **23**(3–4), 130–145 (2008)
24. Rodell, M., Houser, P.R., Jambor, U.E.A., Gottschalck, J., Mitchell, K., Meng, C., Arsenault, K., Cosgrove, B., Radakovich, J., Bosilovich, M., et al.: The global land data assimilation system. *Bull. Am. Meteorol. Soc.* **85**(3), 381–394 (2004)
25. Teague, W.J., Carron, M.J., Hogan, P.J.: A comparison between the Generalized Digital Environmental Model and Levitus climatologies. *J. Geophys. Res. Oceans* **95**(C5), 7167–7183 (1990)
26. Wan, W., Xiao, J., Hong, X., Tan, G.: Parallel implementation and optimization of large scale ocean data assimilation algorithm, pp. 1–10. *CCF* (2018)
27. Xiao, J., Wang, S., Wan, W., Hong, X., Tan, G.: S-enkf: co-designing for scalable ensemble Kalman filter. In: *Proceedings of the 24th symposium on principles and practice of parallel programming*, pp. 15–26. *ACM* (2019)
28. Yan, C., Zhu, J., Tanajura, C.A.S.: Impacts of mean dynamic topography on a regional ocean assimilation system. *Ocean Sci.* **11**(5), 829–837 (2015)
29. Zhang, W., Zhu, X., Zhao, J.: Implementation of phase domain decomposition parallel algorithm of three-dimensional variational data assimilation. *J. Comput. Res. Dev.* **6**, 1059–1064 (2005)
30. Zupanski, M.: *Theoretical and Practical Issues of Ensemble Data Assimilation in Weather and Climate*, pp. 67–84. Springer, Berlin (2009)

## Affiliations

Junmin Xiao<sup>1,2</sup>  · Guizhao Zhang<sup>3</sup> · Yanan Gao<sup>4</sup> · Xuehai Hong<sup>3</sup> · Guangming Tan<sup>1,2</sup>

Guizhao Zhang  
zhangguizhao17g@ict.ac.cn

Yanan Gao  
gaotunny@126.com

Xuehai Hong  
hxx@ict.ac.cn

Guangming Tan  
tgm@ict.ac.cn

- <sup>1</sup> State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
- <sup>2</sup> University of Chinese Academy of Sciences, Beijing, China
- <sup>3</sup> Institute of Computing Technology, Chinese Academy of Sciences, Beijing, China
- <sup>4</sup> Beijing Institute of Control Engineering, Beijing, China