S-EnKF: Co-designing for Scalable Ensemble Kalman Filter

Junmin Xiao, Shijie Wang, Weiqiang Wan, Xuehai Hong, and Guangming Tan
State Key Laboratory of Computer Architecture, Institute of Computing Technology, Chinese Academy of Sciences
University of Chinese Academy of Sciences
xiaojunmin@ict.ac.cn {wangshijie, wanweiqiang}@ncic.ac.cn {hxh, tgm}@ict.ac.cn

Abstract

Ensemble Kalman filter (EnKF) is one of the most important methods for data assimilation, which is widely applied to the reconstruction of observed historical data for providing initial conditions of numerical atmospheric and oceanic models. With the improvement of data resolution and the increase in the amount of model data, the scalability of recent parallel implementations suffers from high overhead on data transfer. In this paper, we propose, S-EnKF: a scalable and distributed EnKF adaptation for modern clusters. With an in-depth analysis of new requirements brought forward by recent frameworks and limitations of current designs, we present a co-design of S-EnKF. For fully exploiting the resources available in modern parallel file systems, we design a concurrent access approach to accelerate the process of reading large amounts of background data. Through a deeper investigation of the data dependence relations, we modify EnKF's workflow to maximize the overlap of file reading and local analysis with a new multi-stage computation approach. Furthermore, we push the envelope of performance further with aggressive co-design of auto-tuning through tradeoff between the benefit on runtime and the cost on processors based on classic cost models. The experimental evaluation of S-EnKF demonstrates nearly ideal strong scalability on up to 12,000 processors. The largest run sustains a performance of 3x-speedup compared with P-EnKF, which represents the state-of-art parallel implementation of EnKF.

CCS Concepts • **Computing methodologies → Massively parallel algorithms**; • **Applied computing → Environmental sciences**.

Keywords ensemble Kalman filter, parallel implementation, domain localization, data assimilation, scalability

PPoPP '19, February 16–20, 2019, Washington, DC, USA

ACM ISBN 978-1-4503-6225-2/19/02…$15.00
https://doi.org/10.1145/3293883.3295722

1 Introduction

Data assimilation initially developed in the field of numerical weather prediction. It is widely applied to the reconstruction of observed historical data for providing initial conditions of numerical atmospheric [16, 20] and oceanic models [7, 34]. The ensemble Kalman filter (EnKF) is a sophisticated sequential data assimilation method. It applies an ensemble of model states to represent the error statistics of the model estimate [27], and utilizes ensemble integrations to predict the error statistics forward in time [22]. In EnFK, an analysis scheme would operate directly on the ensemble of model states when observations are assimilated [17]. As it is essential for the atmospheric and oceanic physics research to understand dynamic behaviors of the global circulation at increasingly fine resolutions [9], high resolution data assimilation has been developed in recent years. In order to enable high-fidelity assimilation for realistic problems, the study of parallelization for EnKF is becoming an urgent demand.

Recent decades have witnessed the development of parallel methods for the implementation of EnKF. Several parallel local EnKF (L-EnKF) implementations have been proposed [25] based on domain localization with a given radius of influence r. Usually, the parallel assimilation process is performed for each model component in parallel making use of a deterministic formulation of the EnKF in the ensemble space. In this formulation, the unknown background error covariance matrix is estimated by the rank-deficient ensemble covariance matrix which is well-defined in the ensemble space. L-EnKF relies on an assumption that local analysis avoids the impact of spurious correlations, for instance, by considering only small values for r. However, in operational data assimilation, r can be large owing to circumstances such as sparse observational networks and/or long distance data error correlations (i.e., pressure fields). In such cases, the accuracy of L-EnKF can be negatively impacted owing to spurious correlations. In order to improve the accuracy of L-EnKF methods, by using a better estimation of background error correlations based on the modified Cholesky decomposition, an efficient parallel ensemble Kalman filter (P-EnKF) has been proposed, which is also optimized deeply, and represents the state-of-art implementation [23, 24]. Although P-EnKF outperforms in terms of accuracy of model variables, its computational time is close to that of L-EnKF methods

[24]. The main reason is that P-EnKF is designed in the same computational framework used by all L-EnKF methods.

The recent parallel framework for EnKF can be simply described as two phases: obtaining local data and executing local update. The popular optimization method is improving the parallel performance of each phase respectively. It seems difficult to overlap the two phases and consider them as a whole, because the local update does not begin until the local data has been obtained by processors. Although the processor which is the first getting all local data, could first execute the local analysis, the overlap ratio of two phases is small. Since the two phases almost are separate from each other, and more parallel file I/O in the first phase would take longer time significantly with the data density increasing, the parallel scalability of the recent framework suffers from some bottleneck problems [10]. For example, we consider the data assimilation of 0.1° resolution data with 120 samples. The time for file reading dominates the main part of the runtime with the number of processors increasing (Figure 1), which negatively impacts the performance of parallel implementations for EnKF (refer to Section 5.2).

In order to efficiently co-design file reading and local updating, this work would do a deeper investigation of the data structures, and re-design the recent workflow. Based on the fine-grain analysis, we propose S-EnKF: a novel and scalable EnKF that provides efficient scale-up and scale-out for distributed computation on HPC systems. To the best of our knowledge, this is the first study that provides an in-depth analysis of fundamental algorithm level bottlenecks being faced by data assimilation methods like P-EnKF, and highlights novel co-designed solutions to achieve high performance and scalability.

We make the following key contributions:

- Analyze and identify new requirements brought forward by the recent EnKF framework, especially for large amounts of I/O and communication.
- Co-design, implement, and evaluate S-EnKF: a scalable and distributed EnKF.
- Propose a concurrent access approach to speed up the process of file reading and fully exploit the resources available in modern parallel file systems.
- Propose a novel overlapping strategy for data obtaining and local updating via multi-stage computation and helper-thread based communication.
- Co-design and implement an auto-tuning approach for determining the optimal parameters of the overlapping strategy in order to make the most advantage of distributed-memory platforms.
- Evaluate and analyze the performance of the proposed S-EnFK with large data sets, which proves that S-EnFK supports large-scale data assimilation well.

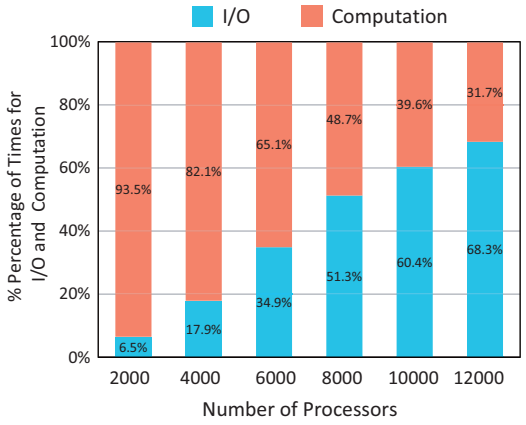

Figure 1. Percentage of times for I/O and computation in P-EnKF.

2 Preliminaries

In this section, we provide an overview of different parallel implementations of EnKF, and a detailed description of the recent EnKF framework.

2.1 Overview of EnKF

For the data assimilation, EnKF gives the solution by solving the following equation

$$\mathbf{X}^a = \mathbf{X}^b + \delta\mathbf{X}^a \in \mathbb{R}^{n\times N}, \tag{1}$$

where n is the number of model components, and N is the ensemble size. $\mathbf{X}^a$ is the analysis result. $\mathbf{X}^b$ is a given background ensemble which is taken from the long-time model integration, and can be represented as

$$\mathbf{X}^b = [\mathbf{X}^{b[1]}, \mathbf{X}^{b[2]}, \cdots, \mathbf{X}^{b[N]}] \in \mathbb{R}^{n\times N}, \tag{2}$$

and $\mathbf{X}^{b[i]} \in \mathbb{R}^{n\times 1}$ is the i-th ensemble member. $\delta\mathbf{X}^a$ is known as the analysis increment. In general, the analysis increment of EnKF reads

$$\delta\mathbf{X}^a = \mathbf{B}\mathbf{H}^T \cdot [\mathbf{R} + \mathbf{H}\mathbf{B}\mathbf{H}^T]^{-1} \cdot [\mathbf{Y}^s - \mathbf{H}\mathbf{X}^b] \in \mathbb{R}^{n\times N}, \tag{3}$$

where $\mathbf{H} \in \mathbb{R}^{m\times n}$ is the linear observational operator, m is the number of observed components, and $\mathbf{R} \in \mathbb{R}^{m\times m}$ is the estimated data error covariance matrix. The superscript T denotes matrix transpose. $\mathbf{Y}^s \in \mathbb{R}^{m\times N}$ is the matrix of perturbed observation with data error distribution $N(\mathbf{0}_m, \mathbf{R})$. $\mathbf{B}$ is the background error covariance matrix which can be estimated by

$$\mathbf{B} = \frac{\mathbf{U}\mathbf{U}^T}{N-1}, \text{ and } \mathbf{U} = \mathbf{X}^b - \overline{\mathbf{x}}^b \otimes \mathbf{1}_N^T \in \mathbb{R}^{n\times N}, \tag{4}$$

where $\overline{\mathbf{x}}^b$ is the ensemble mean, $\mathbf{1}_N \in \mathbb{R}^N$ is a vector whose N components are all ones, and $\otimes$ denotes the outer product of two vectors. If $\hat{\mathbf{B}}^{-1}$ is an approximation of $\mathbf{B}^{-1}$, then the equation (3) is equivalent to

$$\delta\mathbf{X}^a = [\hat{\mathbf{B}}^{-1} + \mathbf{H}^T\mathbf{R}^{-1}\mathbf{H}]^{-1} \cdot \mathbf{H}^T \cdot \mathbf{R}^{-1} \cdot [\mathbf{Y}^s - \mathbf{H}\mathbf{X}^b]. \tag{5}$$

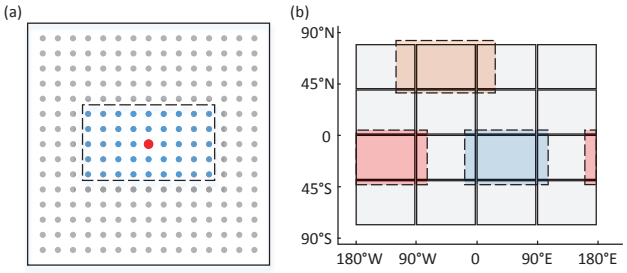

Figure 2. Localization of EnKF.

2.2 Parallel Implementation of EnKF

Localization is commonly used in the context of data assimilation in order to mitigate the impact of spurious correlations in the assimilation process. In general, two forms of localization methods are used: covariance matrix localization and domain localization, both have proven to be equivalent [28]. In practice, covariance matrix localization can be very difficult owing to the explicit representation in memory of the ensemble covariance matrix. On the other hand, domain localization methods avoid spurious correlations by only considering observations within a given radius of influence r. In the 2-dimensional case, each model component is surrounded by a local box of dimension $(2\xi + 1, 2\eta + 1)$ which contains the scope of a circle with radius r. ξ may not be equal to η, since the spacing between two lattice points is different along longitude and latitude directions in a $n_x \times n_y$ mesh with $n_x \gg n_y$ [14]. The information within the scope of the local box, such as observed components and background error correlations, is used in the assimilation process, while the information outside the local box is discarded. In Figure 2(a), a local box for an influence scope with $r = 10$ km is shown (here, $\xi = 4, \eta = 2$). The red grid point is the one to be assimilated, blue points are used in the assimilation process, while gray points are discarded. This characteristic is the foundation of developing parallel implementations of EnKF.

Based on the localization, the domain decomposition is proposed in order to reduce the dimension of the data assimilation problem. At the beginning, the $n_x \times n_y$ latitude-longitude mesh ($n = n_x \times n_y$) is split according to n_s non-overlapping sub-domains along the longitude and latitude directions. Without loss of generality, we assume that there are n_{sdx} and n_{sdy} sub-domains along the longitude and latitude directions respectively, and n_x (or n_y) is a multiple of n_{sdx} (or n_{sdy}). Hence, $n_s = n_{sdx} \times n_{sdy}$ is valid, and the total number of model components at each sub-domain is $n_{sd} = \frac{n_x n_y}{n_{sdx} n_{sdy}}$. Since the update on each point needs the information in a local box (Figure 2(a)), each sub-domain needs some additional model grid points for local analysis. We define the expansion $\bar{D}_{i,j}$ of a sub-domain $D_{i,j}$ as the point set which includes the sub-domain and all additional points needed for local analysis (Figure 2(b)). It means that an expansion contains all data for local assimilation at a sub-domain. Denote $\bar{\mathbf{X}}^{b[k]}_{[i,j]}$ as the data of a background ensemble member $\mathbf{X}^{b[k]}$ on $\bar{D}_{i,j}$ ($k = 1, 2, \cdots, N$). If we consider the data assimilation at a sub-domain $D_{i,j}$, the local analysis reads

$$\begin{aligned}\mathbf{X}^a_{[i,j]} &= \mathbf{P}_{i,j} \cdot [\bar{\mathbf{X}}^b_{[i,j]} + [\hat{\mathbf{B}}^{-1}_{[i,j]} + \mathbf{H}^T_{[i,j]}\mathbf{R}^{-1}_{[i,j]}\mathbf{H}_{[i,j]}]^{-1} \\ &\quad \cdot \mathbf{H}^T_{[i,j]} \cdot \mathbf{R}^{-1}_{[i,j]} \cdot [\mathbf{Y}^s_{[i,j]} - \mathbf{H}_{[i,j]}\bar{\mathbf{X}}^b_{[i,j]}]], \qquad (6)\end{aligned}$$

where $\mathbf{X}^a_{[i,j]} \in \mathbb{R}^{n_{sd} \times N}$ is the analysis result at the sub-domain $D_{i,j}$. $\mathbf{H}_{[i,j]} \in \mathbb{R}^{\bar{m}_{sd} \times \bar{n}_{sd}}$ is the observational operator on $\bar{D}_{i,j}$, and $\bar{n}_{sd} = (\frac{n_x}{n_{sdx}} + 2\xi) \cdot (\frac{n_y}{n_{sdy}} + 2\eta)$, and $\bar{m}_{sd}$ is the number of observed components in $\bar{D}_{i,j}$. $\mathbf{Y}^s_{[i,j]} \in \mathbb{R}^{\bar{m}_{sd} \times N}$ is the subset of perturbed observations. $\hat{\mathbf{B}}^{-1}_{[i,j]} \in \mathbb{R}^{\bar{n}_{sd} \times \bar{n}_{sd}}$ is the local inverse estimation of the background error covariance matrix. $\mathbf{R}_{[i,j]} \in \mathbb{R}^{\bar{m}_{sd} \times \bar{m}_{sd}}$ is the local data-error covariance information. $\mathbf{P}_{i,j} \in \mathbb{R}^{n_{sd} \times \bar{n}_{sd}}$ is a matrix for projecting the model components at $\bar{D}_{i,j}$ into $D_{i,j}$. In the real implementation, $\mathbf{P}_{i,j}$ does not need to be constructed, and we only need to focus on update of model components at the sub-domain $D_{i,j}$. For each sub-domain, the local analysis (6) is computed, and then all results are mapped back onto the global domain.

In the parallel implementation of EnKF [4], the model state is divided according to the number of sub-domains. Next, the background ensemble, the observed components, the observation operator, the estimated data error correlations at each sub-domain need to be read from file systems. After all local information on $\bar{D}_{i,j}$ is obtained by the processor for data assimilation on $D_{i,j}$, the local analysis (6) would be executed.

2.3 Parallel EnKF Framework

To the best of our knowledge, the recent parallel implementations of EnKF follow the same framework. The general parallel framework for EnKF can be simply described as two phases: obtaining local data and executing local update.

For obtaining local data, there are two alternative design choices: the first is to choose a single processor for reading data and scattering them to other processors, and the second is to use all processors to obtain local data from disks parallelly. When the resolution data is not high and the number of background ensemble members is not large, the first approach is usually used. However, on the massive computing platforms, the second approach is considered at first, because it can be optimized for parallel file systems where parallel file reading from disks can be faster than exchanging data using MPI-level communication.

For executing local update, the local assimilation process is carried out in parallel without the need of inter-processor communication. In this phase, the most time-consuming part is performing matrix inversion which is usually achieved

by using Cholesky decomposition [5, 8]. In order to accelerate the local update, some efficient modified Cholesky decompositions are designed in order to obtain approximate estimators of the inverse background error covariance matrix [23, 29]. In general, there are good linear solvers in the current literature, and some of them are well-known and used in operational data assimilation such as LAPACK [1] and CuBLAS [6]. Compact representation of matrices can be used as well, in order to exploit the structures of $\hat{\mathbf{B}}^{-1}_{[i,j]}$ in terms of memory allocation.

3 Challenges for Designing Scalable EnKF

In this section, we elaborate specific challenges, requirements, and issues that need to be addressed in order to design a scalable EnKF on distributed-memory systems.

3.1 Parallel Reading

HPC systems usually use a parallel file system and storage nodes. In order to limit the I/O overhead in reading large size datasets available in the data assimilation, the general framework needs to be redesigned with parallel reading capabilities to take advantage of parallel file systems like Lustre [18]. On the parallel computing platforms at most of supercomputing centers, all files are distributed in several storage nodes, and the users can not exactly know which node stores a given file. Hence, it is difficult to customize an I/O solution for file reading on the system level. Consequently, it becomes important to consider the parallel I/O on the algorithm level. One of the challenges in developing a scalable EnKF is feeding the data efficiently to all the processors for local analysis. For the L-EnKF, a single reader processor communicates the data to the other processors, which can not make full use of parallel file systems. In order to parallelly read a background ensemble member which is stored as an independent file, P-EnKF uses all processors to access different parts of a file simultaneously (Figure 3). When the number of processors becomes large, it is inevitable for processors to line up for accessing data, because disks are not able to support unlimited processors to access at the same time. Moreover, as the files (corresponding to different background ensemble members) are read one after another in recent implementations, the tremendous increase in the number of files may pose additional restrictions on the scalability of parallel EnKF. In conclusion, it is necessary to design an efficient parallel data reading mechanism.

3.2 Exploiting Overlap of File Reading and Local Updating

The recent EnKF framework is both I/O and computation intensive. Hence, in order to scale out such framework, it is mandatory to overlap the computation process with I/O. Indeed, the data assimilation is orchestrated in clearly marked phases like data reading and local updating. The issue with

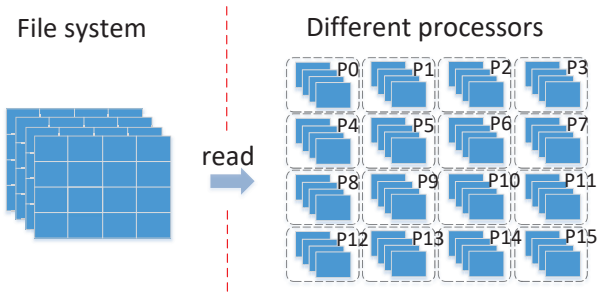

Figure 3. Block reading approach.

Figure 4. Recent workflow for parallel EnKFs.

marked phases that block for the completion of the previous phase is the sequential nature of workflow (Figure 4). In the recent parallel implementations of EnKF, each processor has to obtain all local background ensemble members before it executes the local analysis. Thus, the I/O and computation processes are completely separate. Such separation of file reading and local analysis phases severely limits the efficiency and scalability of parallel implementations of EnKF. In order to alleviate such limitations, a total redesign of the workflow to bring overlap between different phases is required. Such redesign requires: (1) an in-depth analysis of the workflow and its different phases, (2) a fine-grain restructuring of the coarse-grain phases thereby exposing potential for overlap, and (3) an exploitation of new communication ways to perform computation/communication overlap.

4 Co-designs For S-EnKF

We now describe how we tackle the above challenges in designing S-EnKF. Without loss of generality, we assume that the numbers of processors used along the longitude and latitude directions are equal to n_{sdx} and n_{sdy} respectively, and one processor is just responsible for local analysis at only one sub-domain.

4.1 Efficient Parallel Reading Mechanism

For the local analysis (6), $\mathbf{X}^b \in \mathbb{R}^{n\times N}$, $\mathbf{H} \in \mathbb{R}^{m\times n}$, $\mathbf{R} \in \mathbb{R}^{m\times m}$ and $\mathbf{Y}^s \in \mathbb{R}^{m\times N}$ have to be obtained from the file system. Since $N < m \ll n$ in real applications, the sizes of $\mathbf{R}$ and $\mathbf{Y}^s$ are small, and they can be read by each processor directly. Although the size of $\mathbf{H}$ seems larger than that of $\mathbf{X}^b$, the observational operator $\mathbf{H}$ can be constructed from some limited observational data which only need to be read from disk. Hence, I/O cost for $\mathbf{H}$ is small, and obtaining $\mathbf{H}$ is fast. With n and N increasing, the data size of $\mathbf{X}^b$ becomes hundreds

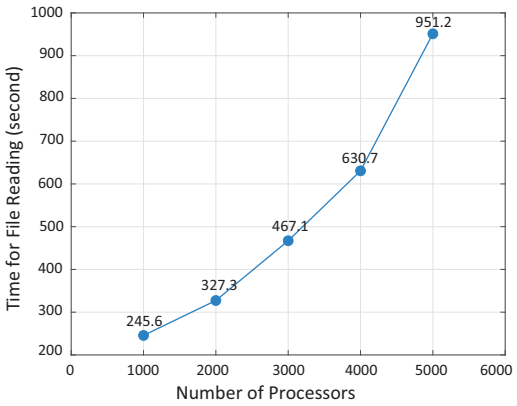

Figure 5. Time for the file reading using the block reading approach. Here, $n_{sdy} = 10$ is fixed, and n_{sdx} increases from 100 to 500.

of *GB*. Therefore, the key to design efficient parallel reading mechanism is how to efficiently read $\mathbf{X}^b$.

4.1.1 Defects of Block Reading Approach

In order to optimize file access operations, we investigate the recent choice which is to use all processors as the data readers for parallel file access with no MPI-level communication (Figure 3). This option is suitable for parallel file systems like Lustre where parallel reading from disk can be faster than exchanging data using MPI-level communication. However, with the improvement of data resolution and the increase in the number of processors involved in data assimilation, a lot of processors would access the same disk simultaneously, which would lead to many processors waiting for the disk resource to become available. Therefore, this approach could not make full use of the concurrent read-write performance of the file system.

On the other hand, due to $n = n_x \times n_y$, each background ensemble member $\mathbf{X}^{b[k]} \in \mathbb{R}^{n\times 1}$, a 1-dimensional vector, can be essentially seen as a 2-dimensional tensor $\mathbf{X}^{b[k]} \in \mathbb{R}^{n_x \times n_y}$ which is contiguously stored in disk according to row priority. From Figure 3, each processor reads a local block $\mathbf{X}^{b[k]}_{[i,j]}$ from the tensor $\mathbf{X}^{b[k]}$. Since each file is split into n_{sdx} parts along longitude (row) direction, any local block could not be read continuously. Further, the number of disk addressing operations in the block reading approach would increase linearly with the number of longitudinal subdivisions increasing, which is $O(n_y \times n_{sdx})$. For instance, Figure 5 presents the time for the block reading approach accessing 100 background ensemble members with different number of processors. It is obvious that the time of this reading approach increases almost linearly with n_{sdx} enlarging.

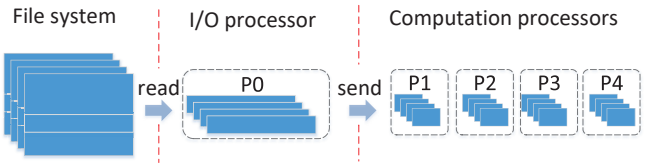

Figure 6. Bar reading approach.

4.1.2 From Block Reading Approach to Bar Reading Approach

Even though the block reading approach can be efficient for small scale and especially for small data sets like atmospheric and oceanic data with 1° resolution [12], to push the scalability and efficiency for EnKF, a co-design approach is required.

In order to effectively avoid a large number of processors simultaneously accessing a disk and reduce the overhead for disk addressing during the file reading, the proposed design is the bar reading approach (Figure 6). Firstly, each data of background ensemble member is divided into n_{sdy} bars along latitude direction, and the i-th bar is denoted as $\bar{\mathbf{X}}^{b[k]}_{[i]}$. Each bar consists of a part of each background ensemble member $\mathbf{X}^{b[k]}$, which is contiguously stored in a disk. Secondly, n_{sdy} processors are chosen for parallel reading of data. They are called I/O processors in which each processor is responsible for reading a bar respectively. Thirdly, I/O processors split the data bar into a lot of overlapped small blocks which are $\bar{\mathbf{X}}^{b[k]}_{[i,j]}$ in (6), and send different blocks to different processors. This approach uses MPI-level communication to broadcast data, but the communication process is fast due to the size of each block being small. Section 5 highlights that, MPI communication time is comparable with the file reading time (Figure 9).

In the bar reading approach, the number of processors for reading data from disks can be chosen according to the maximum disk concurrency of a parallel file system. It is worth mentioning that the bar reading enables each I/O processor to access the contiguous data $\bar{\mathbf{X}}^{b[k]}_{[i]}$ in the disk with only one disk addressing operation.

4.1.3 Concurrent Access Approach Based on Bar Reading

The bar reading approach above mainly focuses on reading one file efficiently, while files are accessed one after another. As shown in Figure 6, a lot of background ensemble members (files) have to be obtained before local analysis starts. All background ensemble members are distributed in the file system, where two different files may be stored in either the same disk or two physical disks. Hence, in the co-design for a scalable EnKF, it is necessary to consider how to efficiently read several files at the same time, which may be stored in different disks with a high probability. The motivation is to

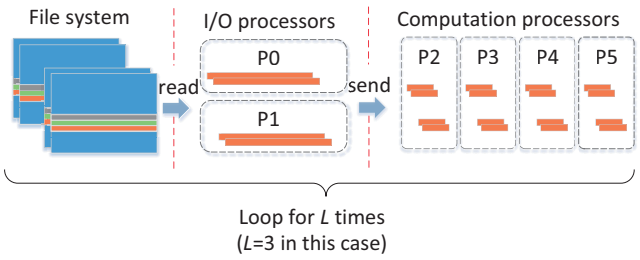

Figure 7. Concurrent access approach for Multi-stage computation.

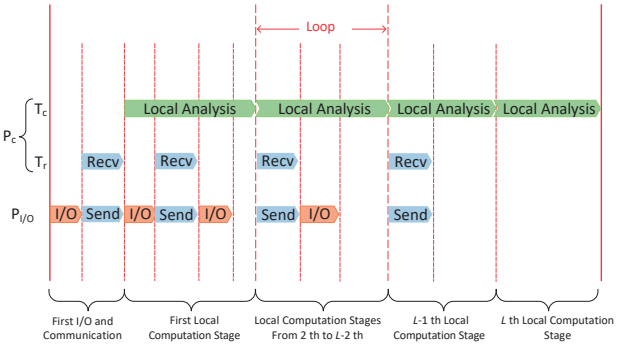

Figure 8. Multi-stage computation strategy.

fully exploit the potential of the concurrent I/O performance supported by disks in a parallel file system.

For reading N background ensemble members, the proposed concurrent access approach is as follows. Firstly, similar to the bar reading approach, several I/O processors are chosen for reading data. Next, the I/O processors are assigned to n_{cg} concurrent groups which are responsible for accessing n_{cg} different files simultaneously. The I/O processors in the same group read the same file at once. Finally, each group only needs to read N/n_{cg} files.

4.2 Multi-stage Computation for Maximal Overlap

As discussed above, the concurrent access approach can effectively improve the data reading process, but it also brings additional communication overhead. Hence, the workflow has three main phases which are file reading, data communication and local analysis. It becomes important for the development of S-EnKF to overlap the local analysis with file reading and data communication.

In order to efficiently co-design the three main phases, we would do a deeper investigation of the data structures and re-design the workflow. Next, since the coarse-grain workflow in the recent framework does not support for the designing of overlapping strategies, we move from a coarse-grain workflow with mainly three successive phases to a fine-grain workflow with multiple stages. Further, we propose a new workflow which could overlap local analysis with file reading and data communication. It is worthwhile to note that the proposed S-EnKF workflow reads data from disks and communicates the block data in an on-demand fashion instead of a coarse-grained approach that performs acquisition of local data in a single operation.

In the coarse-grain workflow of EnKF based on the concurrent access approach, the file reading, data communication and local analysis proceed after the completion of the former one. Before each processor executes the local update, it must obtain the data on the same block of all background ensemble members. This fact implies that it is impossible to overlap computation with I/O and communication operations in the recent framework. However, if we view the procedure above in fine-grain, new insights would be obtained. First of all, we consider the update of one grid point in any sub-domain. Figure 2(a) shows that the update of one grid point only depends on the points within a certain range around it. This fact implies that updating one point in a sub-domain $D_{i,j}$ does not need all data on the expansion $\bar{D}_{i,j}$. Hence, if we plan to update a few of the points in a sub-domain, we only need to read a part of data rather than all information on the expansion. Secondly, if we decompose $D_{i,j}$ into L layers $D'_{i,j,l}$ and update $D'_{i,j,l}$ one after another, then the local analysis on $D_{i,j}$ would be divided into multiple stages. Thirdly, for updating $D'_{i,j,l}$, the file reading and data communication would be adjusted accordingly. The necessary data only are read and sent for the local analysis on $D'_{i,j,l}$ (Figure 7). By this way, a coarse-grain workflow with mainly three successive phases is changed into a fine-grain workflow with multiple stages, which provides an opportunity for designing overlapping strategies.

As shown in Figure 8, we propose a multi-stage on-demand design to overlap local analysis with file reading and data communication. On one hand, to maximize the overlap potential, we use some processors for I/O and the others for local analysis. I/O and computation processors work simultaneously. At the beginning, each domain $D_{i,j}$ is decomposed into L layers $D'_{i,j,l}$ which would be updated one after another ($1 \le l \le L$). By this way, the local analysis on $D_{i,j}$ is also divided into L stages. At the l-th stage, the computation processor with (i, j)-index, executes the local analysis on $D'_{i,j,l}$, and the I/O processors read the data for the $(l + 1)$-th stage using the concurrent access approach. On the other hand, to split the flow between the communication and computation phases in the computation processors, we have offloaded the invocation of the communication with I/O processors, to a helper thread. The helper thread signals the main thread to invoke the local updates in a multi-stage fashion. Figure 8 shows how computation phases hide the reading operation and communication in the multi-stage flow.

Table 1: Notations used in cost models

Variable	Definition
N	Number of background ensemble members (files)
n	Number of model components (gird points)
n_x	Number of gird points along longitude direction
n_y	Number of gird points along latitude direction
n_{sdx}	Number of sub-domains along longitude direction
n_{sdy}	Number of sub-domains along latitude direction
n_{cg}	Number of concurrent groups for file reading
a	Startup time per message
b	Transfer time per Byte for messages
c	Computation cost of local analysis on each gird point
θ	Transfer time per Byte from disk to memory
ξ	Radiu of influence along longitude direction
η	Radiu of influence along latitude direction
L	Number of layers in each domain
h	Volume of data per gird point

4.3 Modeling Cost of Multi-stage Computation

To estimate the cost of the operation, we extend the cost models used in [3, 26, 30] by treating reading data from disk to node-memory similar to the shared-memory copies. Table 1 presents all variables used in cost models. We assume full-duplex mode of communication, i.e., messages going in one-direction do not affect messages in the opposite direction. With this model, the costs of three main procedures in the multi-stage computation strategy can be represented as follows.

(1) Reading data from disks to node-memory: In this phase, each background ensemble member is decomposed into $n_{sdy} \times L$ overlapping small bars, and an I/O concurrent group is responsible for reading small bars from N/n_{cg} files. n_{sdy} processors in the same concurrent group read different parts of a file, and each processor gets one small bar at a time. The cost of this phase is

$$T_{read} = ((\frac{n_y}{n_{sdy} \cdot L} + 2\eta) \cdot n_x \cdot h \cdot \frac{N}{n_{cg}} \cdot \theta) \cdot \log(n_{cg} \cdot n_{sdy}). \quad (7)$$

(2) Communication: In this phase, each I/O processor sends n_{sdx} starting addresses of different parts in the small bar data to n_{sdx} different computation processors. After receiving the communication requests, each helper thread of computation processors would receive n_{cg} block data from n_{cg} I/O processors in different concurrent groups, and the size of each block data is $(\frac{n_y}{n_{sdy} \cdot L} + 2\eta) \times (\frac{n_x}{n_{sdx}} + 2\xi) \times \frac{N}{n_{cg}}$. Assume that all processors are at equal status in the distributed computing platform, and we do not distinguish the processors within a computation node and the processors across nodes. The cost of this phase is

$$\begin{aligned} T_{comm} &= n_{sdx} \log(n_{cg} + 1) \cdot (a + b(\frac{n_y}{n_{sdy} \cdot L} + 2\eta) \\ &\quad \cdot (\frac{n_x}{n_{sdx}} + 2\xi) \cdot \frac{N}{n_{cg}} \cdot h). \end{aligned} \quad (8)$$

(3) Computation: In this phase, each computation process performs the local analysis on the locally gathered data. Each grid point is updated one after another. Assume that time for updating any point is the same. The cost for a local analysis on $D'_{i,j,l}$ is

$$T_{comp} = c \cdot \frac{n_y}{n_{sdy} \cdot L} \cdot \frac{n_x}{n_{sdx}}. \quad (9)$$

Combining all phases above, we can compute the total cost of our overlapping strategy

$$T_{total} = T_{read} + T_{comm} + L \cdot T_{comp}. \quad (10)$$

4.4 Auto-Tuning for Optimal Parameters

Based on the model (10), a naive approach to determine the optimal parameters n_{sdx}, n_{sdy}, L and n_{cg} is minimizing the total time of our overlapping strategy. However, it is not hard to check that T_{total} is decreasing with the increase of n_{cg}, which means that if the more processors are used, the total runtime is smaller. In fact, the number of processors for reading and updating data is our resource cost for executing EnKF, and the cost is not infinite in the real application. Hence, the proposed approach is considering how to gain the balance between the cost on processors and the benefit on the runtime.

Set $C_1 = n_{cg} \cdot n_{sdy}$ and $C_2 = n_{sdx} \cdot n_{sdy}$, which are the numbers of processors for reading and updating data. They are also called the costs for file reading and local analysis respectively. In order to make it easier to compare the multi-stage computation strategy with the original one, we fix the cost C_2 for the local analysis, because the local analysis is the main operation of EnKF. When C_2 is fixed, it is easy to check that $L \cdot T_{comp}$ becomes a constant. It means that the maximal benefit on the runtime would come from $T_{read} + T_{comm}$ by appropriately paying a cost C_1. Hence, for a fixed C_2, the key to design an auto-tuning for optimal parameters is how to choose an economic cost C_1 for file reading.

First of all, before finding the economic cost C_1, we consider the following optimization problem

$$\min_{n_{sdx}, n_{sdy}, L, n_{cg}} T_1 := T_{read} + T_{comm}, \quad (11)$$

such that

$$n_{cg} \cdot n_{sdy} = C_1 \text{ and } n_{sdx} \cdot n_{sdy} = C_2, \quad (12)$$

where two costs C_1 and C_2 are given. It is easy to solve the problem (11). Our proposed algorithm is shown in Algorithm 1, which searches for the minimum of the objective function by traversing the complete search space.

Next, when the cost C_1 changes, the minimal value of T_1 would vary accordingly (refer to Figure 12). Hence, it is important to determine which cost C_1 is the most economic one based on the different minimal values of T_1 in different cases. Assume that C_1 is taken from a set $\{c_1^1, c_1^2, \cdots, c_1^M\}$ where $c_1^m < c_1^{m+1}$. For $C_1 = c_1^m$, we denote the corresponding minimal value of T_1 as t_1^m. Due to $c_1^m < c_1^{m+1}$, $t_1^m \geq t_1^{m+1}$ is

valid. For $1 \leq m < M$, we define an earnings rate as

$$r_m = \frac{t_1^m - t_1^{m+1}}{c_1^{m+1} - c_1^m}. \tag{13}$$

It is easy to check that $r_m > 0$ and r_m decreases monotonously with m increasing. According to this fact, the proposed condition for determining the most economic cost is that, if

$$r_m < \varepsilon, \tag{14}$$

where ε is a given positive number, then the cost c_1^m is chosen. It means that if more cost can not provide significant benefit any more, we choose the current cost. Based on Algorithm 1 with $C_1 = c_1^m$, we can obtain the corresponding optimal parameters $\hat{n}_{sdx}$, $\hat{n}_{sdy}$, $\hat{L}$ and $\hat{n}_{cg}$.

Finally, we denote n_p as the total number of processors. Based on the way above, if we choose $C_2 = k$ from $k = 1$ to $k = n_p$, we can obtain the corresponding economic cost $C_1 = \hat{c}_1^k$ and optimal parameters $n_{sdx} = \hat{n}_{sdx}^k$, $n_{sdy} = \hat{n}_{sdy}^k$, $L = \hat{L}^k$ and $n_{cg} = \hat{n}_{cg}^k$ in different cases. Substituting $\hat{n}_{sdx}^k$, $\hat{n}_{sdy}^k$, $\hat{L}^k$ and $\hat{n}_{cg}^k$ into the formula (10), we find the minimal T_{total}, and the corresponding parameters are the optimal choice.

In conclusion, Algorithm 2 presents the auto-tuning approach to determine the optimal parameters for the multi-stage computation strategy. The effect of this proposed approach is minimizing the time for first I/O and communication procedures with a suitable cost, and maximally overlapping the following reading and communication with the local updating.

Algorithm 1 Solver for Optimization Model

Initialize $\hat{T}_1 = 0$, $\hat{n}_{sdx} = 1$, $\hat{n}_{sdy} = 1$, $\hat{L} = 1$ and $\hat{n}_{cg} = 1$;
for $j = 1$ to c_1 **do**
  if ($c_1 \% j == 0$) and ($c_2 \% j == 0$) and ($n_y \% j == 0$) **then**
    $k = c_1/j$;
    $i = c_2/j$;
    if ($n_x \% i == 0$) and ($N \% k == 0$) **then**
      for $l = 1$ to n_y/j **do**
        if $(n_y/j) \% l == 0$ **then**
          $n_{sdx} = i$;
          $n_{sdy} = j$;
          $L = l$;
          $n_{cg} = k$;
          Compute $t = T_{read} + T_{comm}$ by (7) and (8);
          if ($\hat{T}_1 == 0$) or ($t < \hat{T}_1$) **then**
            $\hat{T}_1 = t$;
            $\hat{n}_{sdx} = i$;
            $\hat{n}_{sdy} = j$;
            $\hat{L} = l$;
            $\hat{n}_{cg} = k$;
          end if
        end if
      end for
    end if
  end if
end for
return $\hat{T}_1$, $\hat{n}_{sdx}$, $\hat{n}_{sdy}$, $\hat{L}$ and $\hat{n}_{cg}$;

Algorithm 2 Auto-Tuning for Optimal Parameters

Create vectors t, cs, dx, dy, lev and g whose length being n_p;
Initialize $T_{\min} = 0$;
for $c_2 = 1$ to n_p **do**
  $\hat{T} = 0$;
  $k = 0$;
  for $c_1 = 1$ to $n_p - c_2$ **do**
    Obtain $\hat{T}_1$, $\hat{n}_{sdx}$, $\hat{n}_{sdy}$, $\hat{L}$ and $\hat{n}_{cg}$ by Algorithm 1;
    if ($\hat{T} == 0$ and $\hat{T}_1 > 0$) or ($0 < \hat{T}_1$ and $\hat{T}_1 < \hat{T}$) **then**
      $k = k + 1$;
      $\hat{T} = \hat{T}_1$;
      $t[k] = \hat{T}_1$;
      $cs[k] = c_1$;
      $dx[k] = \hat{n}_{sdx}$;
      $dy[k] = \hat{n}_{sdy}$;
      $lev[k] = \hat{L}$;
      $g[k] = \hat{n}_{cg}$;
    end if
  end for
  for $m = 1$ to $k - 1$ **do**
    $r = \frac{t[m]-t[m+1]}{cs[m+1]-cs[m]}$;
    if $r < \varepsilon$ **then**
      break;
    end if
  end for
  Compute T_{total} by (10) with $n_{sdx} = dx[m]$, $n_{sdy} = dy[m]$, $L = lev[m]$ and $n_{cg} = g[m]$;
  if ($T_{\min} == 0$) or ($0 < T_{\min}$ and $T_{\min} < T_{total}$) **then**
    $T_{\min} = T_{total}$;
    $n_{sdx}^* = dx[m]$;
    $n_{sdy}^* = dy[m]$;
    $L^* = lev[m]$;
    $n_{cg}^* = g[m]$;
  end if
end for
return n_{sdx}^*, n_{sdy}^*, L^* and n_{cg}^*;

5 Performance Evaluation

In this section, we present numerical results on Tianhe-2 supercomputer [31].

5.1 Computational Environment and Data Sets

Tianhe-2 was the world's fastest supercomputer from 2013 to 2015. Each node of Tianhe-2 is equipped with two Intel Ivy Bridge CPUs (24 cores). The 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 500 nodes (12,000 cores) in Tianhe-2, and have shown the performances of different parallel implementations for EnKF respectively on over ten thousand CPUs. To the best of our knowledge, the largest number of CPUs utilized by recent work is $O(1,000)$ [10, 11].

In our tests, 120 background ensemble members ($N = 120$) taken from a long-time ocean model integration with the 0.1° spatial resolution and 30 vertical levels, are used to evaluate the parallel performance of S-EnKF. In the 2-dimensional latitude-longitude mesh, $n_x \times n_y = 3600 \times 1800$. It is worth mentioning that, when n_p processors are utilized in P-EnKF, the number of processors for S-EnKF may be less than or

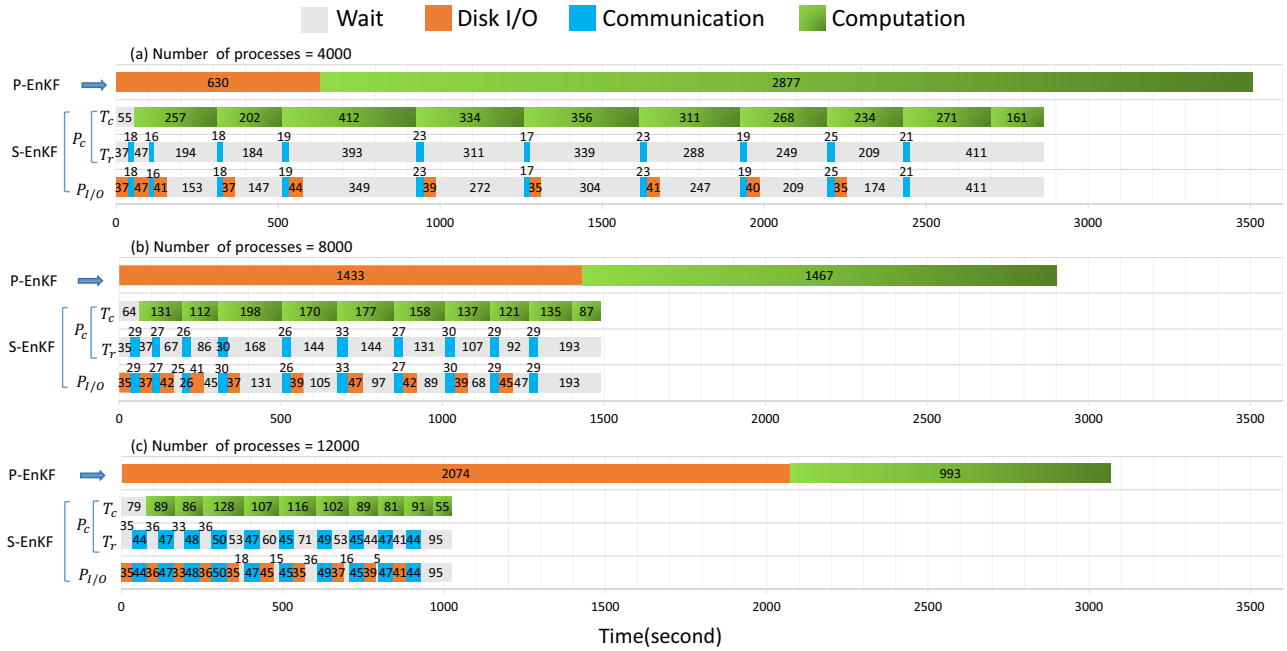

Figure 9. Time for different phases in P-EnKF and S-EnKF.

equal to n_p. The reason is that, for S-EnKF, the total number of processors is the summation of C_1 and C_2, which are determined by Algorithm 2. However, we uniformly denote the number of processors utilized by P-EnKF as the standard for the performance comparison.

5.2 Performance Comparison

Performance comparison focuses on P-EnKF and S-EnKF. P-EnKF represents the state-of-art parallel implementation of EnKF, and is widely used for data assimilation on distributed-memory systems. Our numerical results are presented in Figure 9. It is obvious that P-EnKF is able to scale to about 8,000 cores. However, when the number of processors increases from 8,000 to 12,000, the total runtime does not become shorter due to the time of P-EnKF for file reading increasing significantly. The main reason for this phenomenon has been discussed in Section 4.1.1. However, based on the concurrent access approach, the time of S-EnKF for file reading and data communication only changes slightly with the number of processors increasing. Compared with P-EnKF, S-EnKF achieves 3x speed-up on 12,000 cores. For S-EnKF, the file reading and data communication are hidden well, which results in a good scalability.

As seen from Figure 9, it is obvious that the time for waiting decreases as the number of processors increases. When the number of processors reaches 12,000, the wait time becomes very short, especially for I/O processors. This fact indicates that S-EnKF is able to take full advantage of the parallel performance of the cluster for assimilating the data with 0.1° resolution. It means that our co-design is optimal.

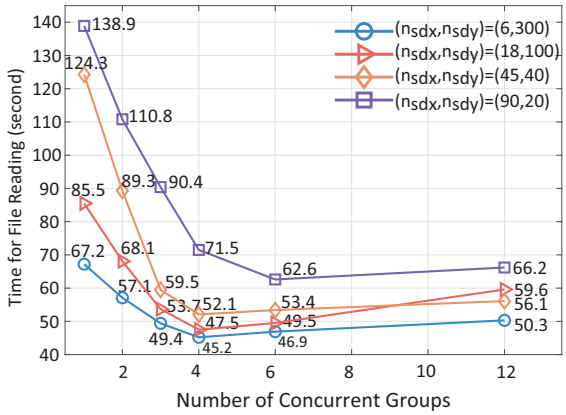

Figure 10. Time for reading 120 background ensemble members with the concurrent access approach.

5.3 Evaluation of Various S-EnKF Co-designs

Concurrent Access Approach: Figure 10 shows the data reading time varying with the number of concurrent groups increasing. When $n_{cg} < 4$, the data reading time decreases monotonously due to the concurrent I/O potential of the file system. As $n_{cg} > 6$, the data reading time changes slightly. The main reason is that, when n_{cg} is large enough, the total I/O bandwidth is fully used. With the number of processors increasing, the time of block reading approach is increasing

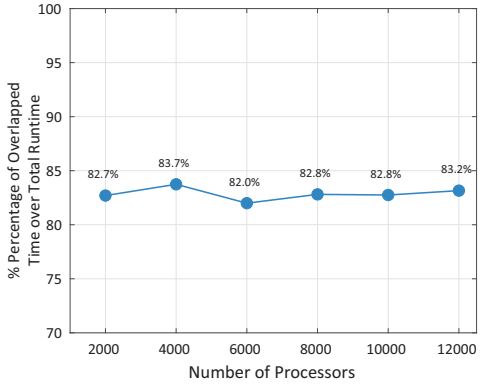

Figure 11. Percentage of the overlapped time over total runtime in S-EnKF.

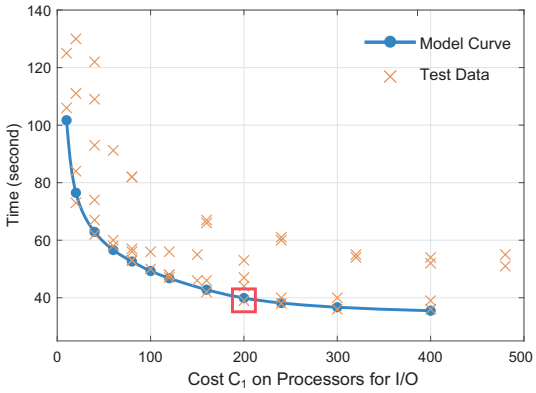

Figure 12. Curve of the minimal value of T_1 and test data with different parameters n_{sdx}, n_{sdy}, L and n_{cg} in the case of $C_2 = 2,000$.

tremendously (Figure 5). However, the time of concurrent access approach is short and controllable, which plays an important role for developing S-EnKF. By the way, the optimal number of concurrent groups, such as 4 or 6 for different cases in Figure 10, would be gained by our auto-tuning approach.

Overlapping Strategy: Define the overlapped time as the time (for waiting, disk I/O and communication) which is overlapped with the time for local computation. Figure 11 shows the percentage of the overlapped time over total runtime in S-EnKF. With the number of processors increasing, the time for file reading and communication changes slightly, while the time for local analysis and the wait time reduce synchronously (Figure 9). Hence, it is reasonable that the percentage of the overlapped time over total runtime is sustained. It means that the multi-stage computation strategy successfully realizes the overlapping of data obtaining and local updating indeed, and the effect would not degrade with the number of processors increasing.

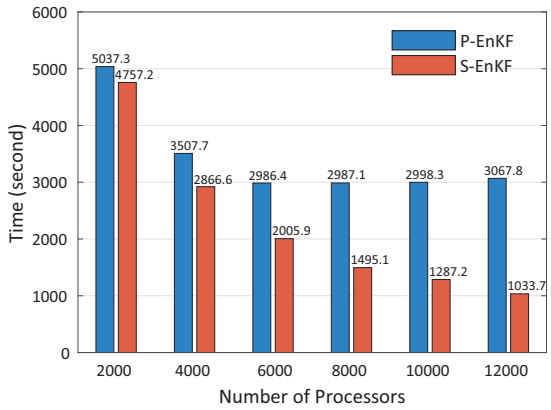

Figure 13. Total runtime of P-EnKF and S-EnKF.

Auto-Tuning Design: Figure 12 presents the minimal value of the objective function T_1 in the optimization model (11) and the corresponding test results in different cases. When the cost C_1 on processors for file reading is fixed, the test results marked by the crosses along vertical direction, are determined by different values of the parameters n_{sdx}, n_{sdy}, L and n_{cg} which satisfy that $n_{sdy} \cdot n_{cg} = C_1$. From Figure 12, it is obvious that the minimal test result with a given cost C_1 is closer to the minimal value of the objective function T_1 than others. Furthermore, we also check that the parameters for the minimal test result and for the minimal value of T_1 are the same. This fact proves that our model is able to reflect the actual performance of the multi-stage computation indeed.

In Figure 12, the square mark highlights the two most economic choices which are determined by the condition (14) based on the test data and the model (11) respectively. It is clear that two economic choices are consistent, which implies that the auto-tuning approach for determining the optimal parameters is effective.

5.4 Primary Highlights

In the scaling tests, we fix the total problem size and increase the number of computing nodes. The total compute time should decrease as more computing nodes are utilized. However, because the computation-communication ratio becomes smaller at a higher node count, which will eventually affect the scalability, ideal scaling results are hard to obtain. Even a small fluctuation in the MPI communication will degrade the performance in the scaling tests [31, 32]. Therefore, successfully overlapping the local computation with file reading and communication operations is a key to achieve expected scaling results.

Figure 13 shows the strong scaling performance of P-EnKF and S-EnKF. As seen from Figure 13, when the number of processors is larger than ten thousand, the runtime of P-EnKF becomes longer with the number of processors increasing.

In contrast, for S-EnKF, nearly ideal strong scaling efficiency is sustained when 12,000 processors are utilized. There is only a very slight loss of scalability in the strong scaling tests, for which the reason can be analyzed from Figure 9. In Figure 9, the most parts of phases for file reading and data communication on the I/O processor side are totally hidden behind the computations on the computation processor side. The only part in the algorithm that could not be overlapped is the first file reading and data communication. Although this part only takes a small portion (less than 8%) of the total computing time, it still affects the parallel efficiency at larger processor counts. We would like to point out here that the parallel efficiency of S-EnKF in the strong scaling results are superior to that of P-EnKF in [23, 24], in which file reading is not overlapped with local computations, and the computing capacities from the CPUs are not fully exploited.

6 Related Work

We now compare S-EnKF with prevalent research and development efforts. L-EnKF [13, 33] is one of the most recent efforts in scaling out EnKF. It discusses scaling of data assimilation to multi-MPI clusters with prime focus on the domain localization given a radius of influence r. It uses a single processor for reading background ensemble members one by one and distributing the data to other processors serially, which is not efficient for reading a large number of high resolution data [15, 19]. However, S-EnKF provides an algorithm perspective of new co-designs and highlights strategies to accelerate file reading phases in the data assimilation. S-EnKF considers how to use several processors for reading many files at the same time to take full advantage of the I/O performance of parallel file systems. On the other hand, P-EnKF [24] represents the state-of-art parallel implementation for EnKF. It exploits modified Cholesky decomposition for accelerating the local analysis. The parallel block reading approach in P-EnKF is more suitable for large scale problems. However, S-EnKF uses a new concurrent access approach based on parallel bar reading, which can reduce the frequency of disk addressing operations. Furthermore, S-EnKF firstly achieves the overlapping of local analysis with the I/O and communication phases, which improves the scalability of P-EnKF significantly. To the best of our knowledge, the maximum number of processors is $O(1,000)$ in recent reports [2, 11, 21]. However, this work at first discusses the performance of parallel implementations with over ten thousands of processors, which proves that S-EnKF is scalable for the data assimilation on large amounts of data with high resolution.

To estimate the cost of the operation, several cost models have been proposed and wildly applied [3]. To the best of our knowledge, almost all of the related work do not use the cost models to determine the optimal number of processors for reading files and the optimal number of dissections for sub-domain in domain localization, because the more benefit on time can be obtained using the more cost on processors. Considering the tradeoff between the benefit and cost, this work firstly proposes an auto-tuning approach to determine the optimal number of processors for file reading and the optimal parameters for domain localization in S-EnKF, which successfully extends the cost models to a broad set of applications.

7 Conclusion

In this paper, we have tackled the challenge of designing a scalable and distributed EnKF. Using a co-design of concurrent access approach for file reading, we fully exploit the resources available in modern parallel file systems to accelerate reading of background ensemble members. Through multi-stage computation co-design, we achieve maximal overlap of data obtaining and local updating. Furthermore, by considering the tradeoff between the benefit on runtime and the cost on processors based on classic cost models, we push the envelope of performance further with aggressive co-design of auto-tuning. The experimental evaluation demonstrates that S-EnKF has the nearly ideal strong scaling efficiency.

Acknowledgments

The authors would like to thank all anonymous referees for their valuable comments and helpful suggestions. The work is supported by the National Key Research and Development Program of China under Grant No. 2016YFC1401700, and National Natural Science Foundation of China under Grant No.(61802369, 61521092, 91430218, 31327901, 61472395, 61432018) and CAS (QYZDJ-SSW-JSC035).

References

[1] E. Anderson, Z. Bai, J. Dongarra, A. Greenbaum, A. McKenney, J. Du Croz, S. Hammarling, J. Demmel, C. Bischof, and D. Sorensen. 1990. LAPACK: A Portable Linear Algebra Library for High-performance Computers. In *Proceedings of the 1990 ACM/IEEE Conference on Supercomputing (Supercomputing '90)*. IEEE Computer Society Press, Los Alamitos, CA, USA, 2–11. http://dl.acm.org/citation.cfm?id=110382.110385

[2] Jeffrey L. Anderson and Nancy Collins. 2007. Scalable Implementations of Ensemble Filter Algorithms for Data Assimilation. *Journal of Atmospheric and Oceanic Technology* 24, 8 (2007), 1452–1463. https://doi.org/10.1175/JTECH2049.1 arXiv:https://doi.org/10.1175/JTECH2049.1

[3] Mohammadreza Bayatpour, Sourav Chakraborty, Hari Subramoni, Xiaoyi Lu, and Dhabaleswar K. (DK) Panda. 2017. Scalable Reduction Collectives with Data Partitioning-based Multi-leader Design. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis (SC '17)*. ACM, New York, NY, USA, Article 64, 11 pages. https://doi.org/10.1145/3126908.3126954

[4] Alexander Bibov and Heikki Haario. 2017. Parallel implementation of data assimilation. *International Journal for Numerical Methods in Fluids* 83, 7 (2017), 606–622. https://doi.org/10.1002/fld.4278 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/fld.4278

[5] Peter J. Bickel and Elizaveta Levina. 2008. Regularized estimation of large covariance matrices. *Ann. Statist.* 36, 1 (02 2008), 199–227. https://doi.org/10.1214/009053607000000758

[6] Pozo R Blackford L S, Petitet A. 2002. An Updated Set of Basic Linear Algebra Subprograms (BLAS). *ACM Trans. Math. Softw.* 28, 2 (June 2002), 135–151. https://doi.org/10.1145/567806.567807

[7] Alexandre A. Emerick and Albert C. Reynolds. 2013. Ensemble smoother with multiple data assimilation. *Computers and Geosciences* 55 (2013), 3–15. https://doi.org/10.1016/j.cageo.2012.03.011

[8] Geir Evensen. 2003. The Ensemble Kalman Filter: theoretical formulation and practical implementation. *Ocean Dynamics* 53, 4 (01 Nov 2003), 343–367. https://doi.org/10.1007/s10236-003-0036-9

[9] Kevin Hamilton and Wataru Ohfuchi. 2008. *High Resolution Numerical Modelling of the Atmosphere and Ocean.* Springer.

[10] P. L. Houtekamer, Bin He, and Herschel L. Mitchell. 2014. Parallel Implementation of an Ensemble Kalman Filter. *Monthly Weather Review* 142, 3 (2014), 1163–1182. https://doi.org/10.1175/MWR-D-13-00011.1 arXiv:https://doi.org/10.1175/MWR-D-13-00011.1

[11] P. L. Houtekamer and Fuqing Zhang. 2016. Review of the Ensemble Kalman Filter for Atmospheric Data Assimilation. *Monthly Weather Review* 144, 12 (2016), 4489–4532. https://doi.org/10.1175/MWR-D-15-0440.1 arXiv:https://doi.org/10.1175/MWR-D-15-0440.1

[12] M. N. Kaurkin, R. A. Ibrayev, and K. P. Belyaev. 2016. Data assimilation in the ocean circulation model of high spatial resolution using the methods of parallel programming. *Russian Meteorology and Hydrology* 41, 7 (01 Jul 2016), 479–486. https://doi.org/10.3103/S1068373916070050

[13] Christian L. Keppenne. 2000. Data Assimilation into a Primitive-Equation Model with a Parallel Ensemble Kalman Filter. *Monthly Weather Review* 128, 6 (2000), 1971–1981. https://doi.org/10.1175/1520-0493(2000)128<1971:DAIAPE>2.0.CO;2

[14] Christian L. Keppenne and Michele M. Rienecker. 2002. Initial Testing of a Massively Parallel Ensemble Kalman Filter with the Poseidon Isopycnal Ocean General Circulation Model. *Monthly Weather Review* 130, 12 (2002), 2951–2965. https://doi.org/10.1175/1520-0493(2002)130<2951:ITOAMP>2.0.CO;2

[15] Terasaki Koji, Sawada Masahiro, and Miyoshi Takemasa. 2015. Local Ensemble Transform Kalman Filter Experiments with the Nonhydrostatic Icosahedral Atmospheric Model NICAM. *SOLA* 11 (2015), 23–26. https://doi.org/10.2151/sola.2015-006

[16] Fred Kucharski, Franco Molteni, and Annalisa Bracco. 2006. Decadal interactions between the western tropical Pacific and the North Atlantic Oscillation. *Climate Dynamics* 26, 1 (01 Jan 2006), 79–91. https://doi.org/10.1007/s00382-005-0085-5

[17] Y. Liu, A. H. Weerts, M. Clark, H.-J. Hendricks Franssen, S. Kumar, H. Moradkhani, D.-J. Seo, D. Schwanenberg, P. Smith, A. I. J. M. van Dijk, N. van Velzen, M. He, H. Lee, S. J. Noh, O. Rakovec, and P. Restrepo. 2012. Advancing data assimilation in operational hydrologic forecasting: progresses, challenges, and emerging opportunities. *Hydrology and Earth System Sciences* 16, 10 (2012), 3863–3887. https://doi.org/10.5194/hess-16-3863-2012

[18] Lustre. 2017. Parallel File System. (2017). http://lustre.org

[19] Takemasa Miyoshi and Masaru Kunii. 2012. The Local Ensemble Transform Kalman Filter with the Weather Research and Forecasting Model: Experiments with Real Observations. *Pure and Applied Geophysics* 169, 3 (01 Mar 2012), 321–333. https://doi.org/10.1007/s00024-011-0373-4

[20] F. Molteni. 2003. Atmospheric simulations using a GCM with simplified physical parametrizations. I: model climatology and variability in multi-decadal experiments. *Climate Dynamics* 20, 2 (01 Jan 2003), 175–191. https://doi.org/10.1007/s00382-002-0268-2

[21] Lars Nerger. 2004. *Parallel Filter Algorithms for Data Assimilation in Oceanography.* Ph.D. Dissertation. Universität Bremen. http://epic.awi.de/10313/

[22] Lars Nerger and Wolfgang Hiller. 2013. Software for ensemble-based data assimilation systems–implementation strategies and scalability. *Computers and Geosciences* 55 (2013), 110 – 118. https://doi.org/10.1016/j.cageo.2012.03.026

[23] E. Nino-Ruiz, A. Sandu, and X. Deng. 2018. An Ensemble Kalman Filter Implementation Based on Modified Cholesky Decomposition for Inverse Covariance Matrix Estimation. *SIAM Journal on Scientific Computing* 40, 2 (2018), A867–A886. https://doi.org/10.1137/16M1097031 arXiv:https://doi.org/10.1137/16M1097031

[24] Elias D. Nino-Ruiz, Adrian Sandu, and Xinwei Deng. 2017. A parallel implementation of the ensemble Kalman filter based on modified Cholesky decomposition. *Journal of Computational Science* (2017). https://doi.org/10.1016/j.jocs.2017.04.005

[25] Edward Ott, Brian R. Hunt, Istvan Szunyogh, Aleksey V. Zimin, Eric J. Kostelich, Matteo Corazza, Eugenia Kalnay, D.J. Patil, and James A. Yorke. 2004. A local ensemble Kalman filter for atmospheric data assimilation. *Tellus A: Dynamic Meteorology and Oceanography* 56, 5 (2004), 415–428. https://doi.org/10.3402/tellusa.v56i5.14462 arXiv:https://doi.org/10.3402/tellusa.v56i5.14462

[26] Rolf Rabenseifner. 2004. Optimization of Collective Reduction Operations. In *Computational Science - ICCS 2004*, Marian Bubak, Geert Dick van Albada, Peter M. A. Sloot, and Jack Dongarra (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 1–9.

[27] Vishwas Rao and Adrian Sandu. 2016. A time-parallel approach to strong-constraint four-dimensional variational data assimilation. *J. Comput. Phys.* 313 (2016), 583 – 593. https://doi.org/10.1016/j.jcp.2016.02.040

[28] Pavel Sakov and Laurent Bertino. 2011. Relation between two common localisation methods for the EnKF. *Computational Geosciences* 15, 2 (01 Mar 2011), 225–237. https://doi.org/10.1007/s10596-010-9202-6

[29] Zheqi Shen and Youmin Tang. 2015. A modified ensemble Kalman particle filter for non-Gaussian systems with nonlinear measurement functions. *Journal of Advances in Modeling Earth Systems* 7, 1 (2015), 50–66. https://doi.org/10.1002/2014MS000373

[30] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. 2005. Optimization of Collective Communication Operations in MPICH. *International Journal of High Performance Computing Applications* 19, 1 (2005), 49–66.

[31] W. Xue, C. Yang, H. Fu, X. Wang, Y. Xu, J. Liao, L. Gan, Y. Lu, R. Ranjan, and L. Wang. 2015. Ultra-Scalable CPU-MIC Acceleration of Mesoscale Atmospheric Modeling on Tianhe-2. *IEEE Trans. Comput.* 64, 8 (Aug 2015), 2382–2393. https://doi.org/10.1109/TC.2014.2366754

[32] Chao Yang, Wei Xue, Haohuan Fu, Lin Gan, Linfeng Li, Yangtong Xu, Yutong Lu, Jiachang Sun, Guangwen Yang, and Weimin Zheng. 2013. A Peta-scalable CPU-GPU Algorithm for Global Atmospheric Simulations. In *Proceedings of the 18th ACM SIGPLAN symposium on principles and practice of parallel programming*, Vol. 48. 1–12.

[33] H. Yashiro, K. Terasaki, T. Miyoshi, and H. Tomita. 2016. Performance evaluation of a throughput-aware framework for ensemble data assimilation: the case of NICAM-LETKF. *Geoscientific Model Development* 9, 7 (2016), 2293–2300. https://doi.org/10.5194/gmd-9-2293-2016

[34] Milija Zupanski. 2009. *Theoretical and Practical Issues of Ensemble Data Assimilation in Weather and Climate.* Springer Berlin Heidelberg, Berlin, Heidelberg, 67–84. https://doi.org/10.1007/978-3-540-71056-1_3