Software for Ensemble-based Data Assimilation Systems – Implementation Strategies and Scalability

Lars Nerger*, Wolfgang Hiller

Alfred Wegener Institute for Polar and Marine Research, Am Handelshafen 12, D-27570 Bremerhaven, Germany

Abstract

Data assimilation algorithms combine a numerical model with observations in a quantitative way. For an optimal combination either variational minimization algorithms or ensemble-based estimation methods are applied. The computations of a data assimilation application are usually far more costly than a pure model integration. To cope with the large computational costs, a good scalability of the assimilation program is required. The ensemble-based methods have been shown to exhibit a particularly good scalability due to the natural parallelism inherent in the integration of an ensemble of model states. However, also the scalability of the estimation method – commonly based on the Kalman filter – is important. This study discusses implementation strategies for ensemble-based filter algorithms. Particularly efficient is a strong coupling between the model and the assimilation algorithm into a single executable program. The coupling can be performed with minimal changes to the numerical model itself and leads to a model with data assimilation extension. The scalability of the data assimilation system

*Corresponding author.
Email address: `lars.nerger@awi.de` (Lars Nerger)

is examined using the example of an implementation of an ocean circulation model with the Parallel Data Assimilation Framework (PDAF) into which synthetic sea surface height data are assimilated.

Keywords:
Data Assimilation, Kalman Filter, Parallelization

1. Introduction

Ensemble-based data assimilation algorithms are applied to combine numerical models with observational data for various applications like in meteorology, oceanography, or in the problem of history matching in petroleum research. The algorithms are typically variants of the Ensemble Kalman filter (EnKF, Evensen, 1994; Burgers et al., 1998). The computationally most efficient methods are currently the so-called ensemble-square root Kalman filters (EnSKF). Several of these methods have been developed and classified over the recent years (Bishop et al., 2001; Anderson, 2001; Whitaker and Hamill, 2002; Evensen, 2004; Tippett et al., 2003; Nerger et al., 2011). For stongly nonlinear applications, particle filters are of growing interest (see van Leeuwen, 2009).

All EnSKFs use an ensemble of model state realizations to estimate the error of the model state. A prediction of the error at a future time is computed by integrating each ensemble state independently by the model. The integrations are typically performed until observations are available. At this time, the information from the observations and the ensemble are combined by performing an analysis step based on the Kalman Filter (Kalman, 1960). The quantitative combination of both information sources is computed using the estimated errors of the observations and the ensemble covariance matrix. All ensemble members are updated in the analysis step resulting in an analysis ensemble that represents the new state estimate and the corresponding errors.

Typical ensemble sizes in EnSKF applications are between the order of 10 and 100 states. Because each ensemble state is integrated by the model, the application of an EnSKF is computationally extremely costly. To reduce the execution time of a data assimilation program, the natural parallelism in the ensemble integration can be utilized. As each ensemble state can be integrated independently from the others, all states can be integrated at the same time, if a sufficiently big computer is available. In the analysis step, all ensemble members have to be combined to compute the ensemble error covariance matrix. The analysis step can be parallelized to reduce its execution time. For the original EnKF, parallel implementations were reported by Keppenne (2000); Keppenne and Rienecker (2002) and Houtekamer and Mitchell (2001). The scalability of different ensemble-based Kalman filters was discussed by Nerger et al. (2005b).

The filter algorithms only require a limited amount of information from the model. In general, they can operate entirely on state vectors, rather than individual fields. In the state vector, all relevant fields, or even parameters in the case of parameter-estimation applications, are stored. For the implementation of observation operators, which compute the observed part of a state vector, it is only required to know how a field is stored in the state vector. These properties permit to implement the analysis step of the filter algorithms in a generic way and to call the analysis routine through a generic interface. A generic implementation should allow to use the same filter implementation with different numerical models. Hence, model specific re-coding can be avoided.

The implementation of the analysis step has to be coupled with the model code. Many assimilation systems use today an offline coupling of the model integrations and the analysis step. That is, two separate programs are used to compute the ensemble forecast and the analysis step. While this scheme is flexible, it has a limited efficiency, because the information transfer between the numerical model and the assimilation program computing the analysis is performed using files. In addition, there are start-up costs for each new model integration. An alternative is a direct coupling of the model and the assimilation routines into a single program. Here, one either has the choice to structure the assimilation system such that the time stepping part of a model is implemented as a subroutine or that the assimilation routines are called in the model code without the requirement that the model itself is a subroutine. In this work, the latter implementation strategy is discussed as implemented in the Parallel Data Assimilation Framework (PDAF, Nerger et al., 2005b, available online at http://pdaf.awi.de). This implementation strategy allows to implement a single-program assimilation system with minimal changes to an existing model code. In addition, efficient implementation and parallelization strategies for the assimilation routines performing the analysis step are discussed. PDAF has been used for data assimilation in different applications (e.g., Nerger and Gregg, 2007; Skachko et al., 2008; Rollenhagen et al., 2009; Janjić et al., 2011b).

The paper is structured as follows. In section 2, EnSKF algorithms are reviewed based on the example of the Singular Evolutive Interpolated Kalman filter (SEIK, Pham et al., 1998b; Pham, 2001). The parallelization strategies for the filter algorithm are then discussed in section 3. Section 4 discusses the strategies for coupling the numerical model and the filter algorithm. Finally, the parallel performance is examined using the example of data assimilation into an ocean model in section 5 and conclusions are drawn in section 6.

2. Ensemble Square-root Kalman Filters

EnSKFs estimate the state of the modeled system at some time t_k by the state vector $\mathbf{x}_k$ of size n. The filters assume that the errors in the state are Gaussian distributed. Accordingly, they describe the error of the state by an error covariance matrix $\mathbf{P}_k$. In ensemble-based filters, $\mathbf{x}_k$ and $\mathbf{P}_k$ are represented by an ensemble of N vectors $\mathbf{x}^{(\alpha)}$ $(\alpha = 1, \ldots, N)$ of model state realizations. The state estimate is then given by the ensemble mean

$$\overline{\mathbf{x}}_k := \frac{1}{N}\sum_{i=1}^{N} \mathbf{x}_k^{(i)} \ . \tag{1}$$

The ensemble covariance matrix

$$\mathbf{P}_k := \frac{1}{N-1}\left(\mathbf{X}_k - \overline{\mathbf{X}_k}\right)\left(\mathbf{X}_k - \overline{\mathbf{X}_k}\right)^T \tag{2}$$

is computed from the ensemble matrix

$$\mathbf{X}_k := \left[\mathbf{x}_k^{(1)}, \ldots, \mathbf{x}_k^{(N)}\right] \tag{3}$$

and the matrix of ensemble means $\overline{\mathbf{X}_k} = [\overline{\mathbf{x}_k}, \ldots, \overline{\mathbf{x}_k}]$.

A forecast is computed by integrating the state ensemble with the numerical model until observations become available.

The observations are stored in form of the vector $\mathbf{y}_k$ of size m. The model state is related to the observations by

$$\mathbf{y}_k = \mathbf{H}_k(\mathbf{x}_k^f) + \epsilon_k \tag{4}$$

where $\mathbf{H}_k$ is termed the "observation operator". The vector of observation errors, ϵ_k, is assumed to be a white Gaussian distributed random process with covariance matrix $\mathbf{R}_k$.

The observations are used in the analysis step to compute a corrected ("analyzed") ensemble, which represents the analysis state estimate $\mathbf{x}_k^a$ and covariance matrix $\mathbf{P}_k^a$. The analysis equations are based on the Kalman filter and minimize the error variance for Gaussian error distributions of the state and observations.

2.1. The SEIK Filter

While the ensemble integration is identical for all EnSKFs, the formulations of their analysis step vary. Here, the analysis is exemplified using the SEIK filter (Pham et al., 1998a; Pham, 2001). Comparisons with other filter methods showed that the SEIK filter is computationally very efficient (see Nerger et al., 2005a, 2006). The SEIK filter is very similar to the Ensemble Transform Kalman Filter (ETKF, Bishop et al., 2001) and the results of the numerical experiments would be very similar with the ETKF. We follow the formulation of the SEIK filter used by Nerger et al. (2011), who classified the SEIK filter as an EnSKF. As all operations are performed at the same time t_k, the time index k is omitted.

The analysis step corrects the state estimate and implicitly updates the state covariance matrix from the forecast to the analysis matrix. Subsequently, the forecast ensemble is transformed such that it represents the analysis state estimate and covariance matrix. The analysis covariance matrix be can written as a transformation of the forecast ensemble as

$$\mathbf{P}^a = \mathbf{L}\mathbf{A}\mathbf{L}^T. \tag{5}$$

Here, $\mathbf{L}$ is an $n \times (N-1)$ matrix defined by

$$\mathbf{L} := \mathbf{X}^f \ \mathbf{T} \tag{6}$$

where $\mathbf{T}$ is a matrix of size $N \times (N-1)$ with all entries being equal to $-N^{-1}$ except for those in the diagonal, which are equal to $1 - N^{-1}$. The multiplication with $\mathbf{T}$ results in a subtraction of the ensemble mean from $\mathbf{X}^f$ and the removal of the last column. The matrix $\mathbf{A}$ has size $(N-1) \times (N-1)$ and is defined by

$$\mathbf{A}^{-1} := \rho(N-1)\mathbf{T}^T\mathbf{T} + (\mathbf{H}\mathbf{L})^T\mathbf{R}^{-1}\mathbf{H}\mathbf{L}. \tag{7}$$

The factor ρ with $0 < \rho \leq 1$ is denoted forgetting factor. It leads to an inflation of the estimated forecast state covariance matrix and can stabilize the filter algorithm. The forecast error covariance matrix $\mathbf{P}^f$ can be written in terms of $\mathbf{L}$ and $\mathbf{T}$ as

$$\mathbf{P}^f = (N-1)^{-1}\mathbf{L}\left(\mathbf{T}^T\mathbf{T}\right)^{-1}\mathbf{L}^T. \tag{8}$$

The analysis update of the state estimate is given as a combination of the columns of the matrix $\mathbf{L}$ by

$$\overline{\mathbf{x}}^a = \overline{\mathbf{x}}^f + \mathbf{L}\overline{\mathbf{w}} \tag{9}$$

where the vector $\overline{\mathbf{w}}$ of size $N-1$ is given by

$$\overline{\mathbf{w}} := \mathbf{A}\left(\mathbf{HL}\right)^T \mathbf{R}^{-1}\left(\mathbf{y} - \mathbf{H}\overline{\mathbf{x}^f}\right). \qquad (10)$$

After updating the state estimate according to Eq. (9), the forecast ensemble $\mathbf{X}^f$ is transformed such that it represents $\mathbf{x}^a$ and $\mathbf{P}^a$. The transformation is performed according to

$$\mathbf{X}^a = \overline{\mathbf{X}}^a + \sqrt{N-1}\mathbf{LC}\boldsymbol{\Omega}^T. \qquad (11)$$

Here, $\mathbf{C}$ is a square root of $\mathbf{A}$. It can be computed as the symmetric square root $\mathbf{C}_{sym} := \mathbf{US}^{-1/2}\mathbf{U}^T$ obtained from the singular value decomposition $\mathbf{USV} = \mathbf{A}^{-1}$. $\boldsymbol{\Omega}$ is a matrix of size $N \times (N-1)$ whose columns are orthonormal and orthogonal to the vector $(1,\dots,1)^T$. $\boldsymbol{\Omega}$ can be a random or deterministic matrix with these properties and can be generated using Householder matrices (Pham, 2001; Hoteit et al., 2002).

2.2. Localization of the Analysis

Large-scale applications of EnSKFs commonly employ a localization of the analysis step (see, e.g., Janjić et al., 2011a, for a discussion of different localization methods). For the SEIK filter, typically a combination of domain localization (DL) and observation localization (OL) is used. The DL of the SEIK filter has been discussed by Nerger et al. (2006).

With DL, the analysis step is performed in disjoint analysis domains of the model grid. For example, in a 3-dimensional ocean model an analysis domain can be a single vertical column of grid points. The localization is then performed by taking into account only observations that lie within a pre-defined distance (the influence radius) from the analysis domain. To apply DL, the operations of the analysis and the ensemble transformation are organized in a loop over the disjoint local analysis domains. If all available observations are assimilated for each local analysis domain, the reordered operations result still in the same analysis ensemble as the global analysis discussed in section 2.1.

To formulate the localized analysis equations, let the subscript σ denote a local analysis domain. The domain of the corresponding observations is denoted by the subscript δ. Now, the local SEIK analysis and ensemble transformation can be written analogous to the global analysis (Eqns. 7 – 11) as

$$\begin{aligned}
\overline{\mathbf{x}}^a_\sigma &= \overline{\mathbf{x}}^f_\sigma + \mathbf{L}_\sigma \overline{\mathbf{w}}_\delta, && (12)\\
\overline{\mathbf{w}}_\delta &:= \mathbf{A}_\delta \left(\mathbf{H}_\delta \mathbf{L}\right)^T \mathbf{R}_\delta^{-1} \left(\mathbf{y}_\delta - \mathbf{H}_\delta \overline{\mathbf{x}}^f\right), && (13)\\
\mathbf{A}_\delta^{-1} &:= \rho_\delta (N-1)\mathbf{T}^T\mathbf{T} + (\mathbf{H}_\delta\mathbf{L})^T \mathbf{R}_\delta^{-1}\mathbf{H}_\delta\mathbf{L}\ , && (14)\\
\mathbf{X}^a_\sigma &= \overline{\mathbf{X}^a_\sigma} + \sqrt{N-1}\mathbf{L}_\sigma \mathbf{C}_\delta \mathbf{\Omega}^T && (15)
\end{aligned}$$

where $\mathbf{C}_\delta^{-1}(\mathbf{C}_\delta^{-1})^T = \mathbf{A}_\delta^{-1}$. $\mathbf{H}_\delta$ is the observation operator that projects a global state vector onto the local observation domain. Thus, it combines the operation of a global observation operator with the restriction of the observation vector to the local observation domain. $\mathbf{R}_\delta$ is the observation error covariance matrix on the local observation domain. ρ_δ denotes the local forgetting factor, which can vary for different local analysis domains. The same matrix $\mathbf{\Omega}$ has to be used for each local analysis domain to ensure consistent transformations throughout all local domains.

OL is a common addition to DL. With OL, weight factors are introduced in each local observation error covariance matrix $\mathbf{R}_\delta^{-1}$ such that the influence of observations is reduced with increasing distance from the corresponding local analysis domain (Hunt et al., 2007; Nerger and Gregg, 2007). OL is performed by an element-wise (i.e. Schur or Hadamard) product of $\mathbf{R}_\delta^{-1}$ with a localization matrix $\mathbf{D}$. Hence, equations (13) and (14) are rewritten as

$$\begin{aligned} \overline{\mathbf{w}}_\delta &= \mathbf{A}_\delta \left(\mathbf{H}_\delta \mathbf{L}\right)^T \left(\mathbf{D}_\delta \circ \mathbf{R}_\delta^{-1}\right) \left(\mathbf{y}_\delta - \mathbf{H}_\delta \overline{\mathbf{x}}^f\right), \quad (16) \\ \mathbf{A}_\delta^{-1} &= \rho_\delta (N-1) \mathbf{T}^T \mathbf{T} + \left(\mathbf{H}_\delta \mathbf{L}\right)^T \left(\mathbf{D}_\delta \circ \mathbf{R}_\delta^{-1}\right) \mathbf{H}_\delta \mathbf{L} \,. \quad (17) \end{aligned}$$

Here $\circ$ denotes the Schur product. $\mathbf{D}_\delta$ is usually constructed using correlation functions of compact support. Possible choices are, for example, an exponential decrease or a 5th-order polynomial that mimics a Gaussian function, but has compact support (Gaspari and Cohn, 1999). If $\mathbf{R}_\delta$ is diagonal, $\mathbf{D}_\delta$ can be a diagonal matrix with elements varying according to the distance of an observation from the local analysis domain.

2.3. Numerical Implementation of the Local SEIK Filter

The analysis algorithm of the local SEIK filter is usually implemented in the following steps.

1. **Non-local preparations:** The innovation vector $\mathbf{y} - \mathbf{H}\overline{\mathbf{x}^f}$ and the observed ensemble $\mathbf{HL}$ are computed on the full model grid. The operations involve the reading of observation data from files. The innovation vector and the observed ensemble have to be computed before the loop of local analysis updates is performed, because their values would change inconsistently if they are reinitialized during the sequence of local updates.
2. **Sequence of local analysis updates:** For each local analysis domain, the following computations are performed.

- The local forecast state $\overline{\mathbf{x}}^f_\sigma$ and local matrix $\mathbf{L}_\sigma$ are initialized.
- A search through all observations is performed to find those that lie within the observation influence radius. The available local observations define the localization in the observation operator $\mathbf{H}_\delta$. In addition, the local observation vector $\mathbf{y}_\delta$ is initialized.
- The local state update is computed according to Eq. (12). During these computations, the OL is applied according to the distance of each observation from the local analysis domain.
- The forecast ensemble is transformed according to Eq. (15) to obtain the local analysis ensemble.
- The local analysis state $\overline{\mathbf{x}}^a_\sigma$ and ensemble $\mathbf{X}^a_\sigma$ are used to initialize the corresponding entries in the global state vector and ensemble array.

In the state update and the ensemble transformation several linear algebra operations, like matrix-matrix or matrix-vector products, as well as singular value decompositions are performed. For optimal performance, the functions for matrix-matrix and matrix-vector products provided by the BLAS library and the LAPACK library functions for singular value decompositions can be used. Optimized variants of these libraries are available on most computers.

3. Parallelization Strategies for the Filter Algorithm

As discussed in the introduction, the analysis step of the filter algorithm itself should be parallelized for optimal performance of the full assimilation system of ensemble forecasts and analysis steps. For the parallelization, a distributed memory paradigm is assumed as is utilized with the Message Passing Interface (MPI, Gropp et al., 1994) standard. MPI is commonly used in large-scale numerical models. With distributed memory, only data belonging to a process itself is directly accessible. For data belonging to other processes, an explicit communication of data between the processes has to be performed.

Two parallelization strategies for the filter are possible (see Nerger et al., 2005b). First, mode decomposition can be performed where sub-sets of the ensemble of full states are distributed over all available processes of a parallel program[1]. The alternative is domain decomposition. Here, full ensembles of sub-states representing a physical sub-domain of the model grid are distributed over all processes. Mode decomposition appears to be optimal for the ensemble forecast, as each ensemble member can be integrated independently from the others. However, mode decomposition will lead to a larger amount of data that needs to be exchanged between different processes in the analysis step of the filter algorithm (see, Nerger et al., 2005b). With domain decomposition, each process performs the sequence of local analysis updates only for those local analysis domains that belong to the process. Thus, the sequence of local analysis steps is distributed into shorter sequences that are concurrently computed. In addition, domain decomposition is a standard parallelization strategy in current large-scale numerical models. Accordingly, if the model uses domain decomposition, one can use the same distribution of the model domain in the filter analysis. For this

[1]A process is considered to be a single processing unit of the computer. It can be related to a single processor core for multi-core processors

reason, only the domain-decomposition strategy for the local SEIK filter is discussed here. Nonetheless, the mode decomposition strategy can be additionally applied in the ensemble forecast. The mode decomposition allows to perform the integration of each ensemble state at the same time, while each model instance can use domain decomposition.

With parallelization, the following modifications are performed to the steps in the local SEIK filter:

1. **Non-local preparations:** The observations can be initialized in a distributed way, if each process reads a file that holds only observations that belong to the sub-domain of the process. In the sequence of local analysis updates observations from other processes will be required. This is due to the fact that the extent of local observation domains will reach into sub-domains of other processes. Thus, one has to assemble an observation vector that reaches beyond the sub-domain of the process. A simple way might be to let each process read the global observation vector. However, at least in the application of the observation operator in $\mathbf{H}\overline{\mathbf{x}^f}$ and $\mathbf{HL}$ an information exchange between different processes is required.
2. **Sequence of local analysis updates:** For each local analysis domain, a local analysis is performed according to in section 2.3. If the observation preparation has been performed as described above, all required data is available in the memory of the process. Thus, the local analysis can be performed without any communication of data. This also implies that no changes in the implementation are required when switching from a serial to a domain-decomposed filter implementation.

The sequence of local analysis updates could be further parallelized using shared-memory parallelization with OpenMP. In this case, the sequence of each MPI process would be split into several short sequences that can be executed concurrently.

4. Coupling Model and Filter Algorithm

In ensemble data assimilation, the numerical model is used to compute the ensemble forecast. The model grid defines at which locations the model fields are available. This information is required to implement the observation operator $\mathbf{H}$ in Eq. (4). In addition, the state vector used in the filter algorithm needs to be initialized from model fields and vice versa. These operations require an information transfer between the model and the filter analysis step. The aim for coupling the model with the filter algorithm into an assimilation system should be to obtain a generic data assimilation environment that can be used with different models. This strategy will allow to optimize the assimilation environment and filter algorithms once and to reuse them for different data assimilation applications.

There are two common approaches to connect the implementation of the analysis step to the model:

- **Offline:** Separate programs are used for the model and the analysis step. Files are used to transfer the information on the ensemble states between the model and the analysis step.
- **Online:** The routines for the analysis step and those for the numerical model are compiled into a single program. The information about the

ensemble states is transferred between the model and the analysis step by explicit subroutine calls.

Apart from these possibilities, one might also couple the model and the analysis step by message passing (MPI) communication (see Nerger, 2003, section 8.4) or even by sockets or named pipes.

The offline implementation strategy has the obvious advantage that no changes to the model source code are required to perform the ensemble forecasts. For the integration of each ensemble member, the model program is started with a unique initial state. The program holding the analysis step is executed when the ensemble integration is completed. A clear computational disadvantage of this implementation is that all information transfer between the model and the analysis programs is conducted through files. In addition, for each ensemble member the model is fully restarted. Thus, the model initialization phase is executed for each ensemble state. The required time for the information transfer through files and the repeated model start up can be significant, in particular for large-scale applications that are executed using a large number of processes.

The online implementation is computationally more efficient. Because both the numerical model and the analysis step are compiled into a single program and connected through subroutine calls, the transfer of the ensemble information using files can be avoided. In addition, the repeated model start-up can be avoided, by executing the start-up phase only once and keeping the model field arrays allocated in memory during the analysis step. For an ensemble forecast, it is then sufficient to only re-initialize the model fields and directly start the integration. While the online implementation is more efficient, it involves the explicit combination of the model and the analysis step in the source code. A possible strategy is to implement an assimilation environment that calls the model as a subroutine. This strategy, however, requires to prepare the model code such that it can be called as a subroutine with an interface that is compatible with the assimilation environment. If the model is not yet implemented using a subroutine, e.g. for the time stepping part, it can be complicated to modify the model source code accordingly.

An alternative online implementation strategy is to implement the data assimilation system without requiring that the model is available as a subroutine. Instead, the model code is extended by subroutine calls to the data assimilation framework. This strategy leads to a numerical model with extension for ensemble data assimilation. This implies that the assimilation system can also be executed like the numerical model without extension, but with additional parameter definitions for the assimilation system. An initial form of this implementation strategy was discussed by Nerger et al. (2005b), but it was further refined with the development of the PDAF system.

To exemplify the implementation strategy, a typical structure of a numerical model is assumed as is shown on the left of Figure 1. In the typical structure, the model grid and initial fields are prepared in the initialization phase of the model. Subsequently, the integration is computed in the time stepping phase. After the integration, some post-processing might be performed before the program stops. In case of a parallel model, the parallelization is initialized directly after the start of the program and finalized directly before the program terminates. The different parts of the model might be implemented in the form of subroutines, but this is not required here. The data assimilation program obtained by extending the model code by calls to routines of the data assimilation framework is shown on the right of Fig. 1. The assimilation program with support for ensemble integrations and analysis step can be obtained by inserting four calls to subroutines (indicated in the figure by the prefix "DA_"). In addition, a loop can be inserted around the time stepping part of the model. This "ensemble loop" increases the flexibility when the execution of the assimilation system is configured with parallelization. In general, there can be one or several "model tasks", each of which is computing forecasts of ensemble states. If there is only a single model task, it has to compute the forecast of all states in the ensemble. Thus, when the forecast is computed from time t_a to t_b, the model task has to jump back to time t_a for each new ensemble member that it has to integrate. The implementation has to ensure that the integrations are independent and consistent. For example, if the model uses forcing data, like surface wind stress in an ocean model, it has to be correctly re-initialized. If the program utilizes the parallelism of the ensemble integration, one can configure the execution of the program such that the ensemble size equals the number of model tasks. In this case, all model tasks will only compute forward in time. The consistency of the integration might be easier to achieve with this configuration as, e.g., the forcing never needs to be restored to an earlier time. If the number of model tasks is always equal to the ensemble size, one could also structure the additional subroutine calls in a way that avoids the ensemble loop. The possibilities, however, will depend on the number of processes that are available for the execution of the program. For efficiency, it is important to ensure that all ensemble members can be uniformly distributed over the model tasks.

The inserted subroutine calls initialize the assimilation application, control the ensemble forecasts, and perform the filter analysis step. They can be implemented with the following functionality:

- **DA_init_parallel**: This routine redefines the parallelization of the program, namely the communicators in case of MPI-parallelization. While for the original (forward) model all processes participate in the integration of a single model state, the assimilation system might compute several integrations at the same time. These are performed by the "model tasks", each with a separate set of processes. Next to these process sets, a set of processes that compute the analysis step has to be defined.

- **DA_init**: Following the initialization phase of the model, this routine initializes the assimilation system. Necessary parameters for the assimilation system are defined, like the size of the state vector or the number of ensemble members. In addition, the initial ensemble is read from files. The ensemble is stored in an array that might be distributed over several processes.

- **DA_get_state**: Preceding the integration phase of the model, this routine initializes model fields from a state vector. In addition, it defines the number of time steps ("nsteps" in Fig. 1) over which the forecast is computed. During the forecast phase, it will also define if more ensemble forecasts should be computed, or if the assimilation sequence is completed.

- **DA_put_state**: This routine is called after the integration phase of the model. First, the routine writes the model fields back into the array holding the ensemble of model states. Subsequently, it checks whether the ensemble forecast is completed for the model task to which the calling process belongs. If there are further ensemble states to be integrated by the model task, the routine is exited and the program will jump back to the beginning of the ensemble loop. If the ensemble forecast is completed, the routine for the filter analysis step will be executed. After the analysis step, the program will jump back to the beginning of the ensemble loop.

The routines DA_get_state as well as DA_put_state require the information how the state vectors are related to actual model fields. These routines also utilize information about the available observations. In particular, the temporal availability of observations will define the length of the forecast phase. In addition, the analysis step requires an implementation of the observation operator $\mathbf{H}_k$ as well as the initialization of the vector of observations $\mathbf{y}_k$. The implementation of these functionalities should follow two criteria: First, the assimilation routines listed above should be independent of the definition of the state vector and of the observations. Second, to minimize the changes to the model code, one should avoid to perform the operations directly in the model code. An efficient implementation strategy that fulfills these criteria is the use of call-back routines. These are routines that are called by the assimilation routines in order to perform a specified operation, like the initialization of the observation vector. It is useful to implement the call-back routines in the context of the model. For example, if a model uses

Fortran modules, these modules can be utilized in the call-back routines if they provide, e.g. information about the coordinates of grid points.

To facilitate the implementation of call-back routines for PDAF, they are designed to include only very elementary operations. One example is the initialization of the observation vector. An array in which the observations are stored is allocated within PDAF. Then a call-back routine is called to fill the observation array with the values of the observations. A similar strategy is followed for the observation operator. In this case, a call-back routine is called with a state vector $\mathbf{x}$ and and array for the observed state vector $\mathbf{Hx}$ in its arguments. The task of the call-back routine is then to compute $\mathbf{Hx}$ from $\mathbf{X}$. Finally, also the product $(\mathbf{H}_\delta\mathbf{L})^T\mathbf{R}_\delta^{-1}$, which is required in Eqns. (13) and (14), is computed in a call-back routine. This routine is provided with $(\mathbf{H}_\delta\mathbf{L})^T$ and has to perform the multiplication by $\mathbf{R}_\delta^{-1}$. This implementation strategy allows that both the observation operator and the multiplication by $\mathbf{R}_\delta^{-1}$ can be implemented in their most efficient way. For example, if $\mathbf{R}$ is diagonal, one can easily use this fact to simplify the multiplication with $(\mathbf{H}_\delta\mathbf{L})^T$. If $\mathbf{R}$ is not diagonal, one can implement the product for example by solving of the equation $\mathbf{R}_\delta^{-1}\mathbf{E} = (\mathbf{H}_\delta\mathbf{L})^T$ for $\mathbf{E}$, which can be performed by calling a suitable function of the LAPACK library.

Another call-back routine is provided with the full ensemble array. This routine is a pre/poststep routine that is executed directly before and after the analysis step. It allows to analyze the ensemble, e.g. by computing the ensemble-estimated variances. Another possibility is to check if the ensemble members are physically realistic after the analysis step and to apply necessary corrections if this is not the case. As the data assimilation changes the state based in statistics, one might need to adjust the states to ensure that a stable model integration can be performed. An example is the assimilation of chlorophyll concentrations into a biogeochemical model. As the concentrations are required to be positive, one might need to correct negative concentrations that might result from the analysis step (see, e.g. Ciavatta et al., 2011).

5. Parallel Performance of the Assimilation System

In this section, an oceanographic application is used to examine the parallel performance of an assimilation system that is implemented with the strategy discussed above. The assimilation system is built by combining the Finite Element Ocean Model (FEOM, Danilov et al., 2004; Wang et al., 2008) with the PDAF assimilation framework. The model is configured for the North Atlantic, with a varying resolution of about 20km in the Gulf Stream region and 100km elsewhere, similarly to that used by Danilov and Schröter (2010). Synthetic observations of the sea level elevation are assimilated, which are available at each grid point of the ocean surface. The observation error are assumed to be uncorrelated with a standard deviation of 5cm. The optimal observation influence depends on the ensemble size as well as the model configuration. Typically, it is determined by experimentation with varying radii. As the focus of the experiments is on the parallel performance of the assimilation application, the radius is set to 500km, which should be not too far from the optimal value (see, e.g., Nerger et al., 2006).

A particular property of FEOM is its use of unstructured triangular meshes. Due to this, there are no direct grid point neighbors in longitudinal or meridional directions like in a rectangular grid. In addition, the domain decomposition of the unstructured mesh is obtained by a partitioning software (METIS, Karypis and Kumar, 1999). This partitioning computes sub-domains of approximately equal size and minimum interfaces to neighboring sub-domains. However, the procedure will result in sub-domains of irregular shape. These properties of the model have a direct influence on the implementation of the observation operator. In particular, the distance between two grid points is not directly related to the grid point indices, but one has to use the indices of the grid points to look up their coordinates in coordinate arrays and to compute the distance from these coordinates. Accordingly, one has to search through all observations, when a local observation domain is defined in the LSEIK filter as explained in section 2.3. This search would be easier in a model with a regular grid, as the grid point indices would indicate the distance between the grid points.

The parallel performance will be studied using the quantity denoted "speedup". It is defined by

$$S_{ap} := \frac{T_a}{T_{ap}} \tag{18}$$

where T_a and T_{ap} are the execution time using a processes and ap processes, respectively. An ideal speedup is the factor by which the number of processes is increased. In addition, the discussion will refer to the "parallel efficiency" defined by

$$E_{ap} := \frac{S_{ap}}{p}. \tag{19}$$

A program with ideal speedup will show a parallel efficiency of 1.

5.1. Assimilation with Time-dominant Ensemble Integrations

A typical situation in ensemble data assimilation is that the forecast phase requires most of the computing time. The parallel performance in this situation is assessed here for the case of forecast phases of 10 days length. In this situation, the computing time for the ensemble forecasts is about 94 – 99% of the overall computing time of an assimilation experiment. The program is configured such that the number of model tasks is identical to the ensemble size. Because the ensemble integrations are intrinsically parallel, the time overhead to integrate the ensemble instead of a single state should be negligible. In this experiment, the analysis step is computed by the processes of a single model task. In the analysis step, the same domain decomposition as in the model forecasts is used. Using only a subset of processes for the analysis requires to collect the forecast states from all model tasks on the processes of the single model task used for the analysis. After the analysis step, the states are redistributed to the model tasks. Using the same domain decomposition for the forecasts and the analysis is motivated by the complexity of the communications with an unstructured grid model. If different decompositions were used, it would require a multiple application of the partitioning algorithm. In addition, the complexity of the communication pattern would significantly increase. However, the experiment shows that the additional time required to collect the ensemble states on the processes a single task is very small.

The left panel of Fig. 2 shows the speedup in the case of the assimilation with FEOM. Two cases are shown in which the ensembles consisted of 8 and 64 state vectors. The number of processes per model task was varied between

8 and 64. Thus, the experiments utilized between 64 and 4096 processes[2]. For both ensemble sizes the speedup is almost identical. As expected, the speedup is nearly identical to that of a single integration using the model without data assimilation extension (not shown). The achieved speedup of the assimilation system is about 5.5 for the case of $N = 64$ and 5.9 for $N = 8$. This corresponds to parallel efficiencies between 68% and 74%.

Next to the speedup of the data assimilation system for fixed ensemble sizes, the dependence of the computing time on the ensemble size is important. If the number of processes per model task is kept fixed, the speedup properties of the model will not influence the result. Instead, the parallel properties of the full assimilation system including the communication of model fields between all model tasks and the single task that computes the filter analysis determine the performance. The execution time for different ensemble sizes but fixed number of processes per model task is shown in the right panel of Fig. 2. The time is normalized by the time required to compute the assimilation experiment for an ensemble of 8 states. If 8 processes are used per model task, the total computing time increases by only about 5% if the ensemble size is increased from 8 to 64. For the larger case using 64 processes per model task, the computing time increases on average by

[2]The choice of the minimum number of processes is motivated by the computers used for the experiments. The experiments have been performed on the computers of the North-German Supercomputing Alliance (HLRN). Each compute node contains two 4-core Intel Xeon Gainestown (Nehalem EP) processors. The nodes are connected by an Infiniband network. Only entire compute nodes have be used, because the processor frequency might be reduced if nodes are incompletely used, which would lead to inconsistent timings.

about 12%. However, due to the large number of processes, the computation times in this larger case varied significantly if the same experiment was repeated. The resulting uncertainty is shown in the right panel of Fig. 2 as error bars. Within one standard deviation, the time increase for $N = 64$ can vary between about 6% and 16%. The fluctuations in the computing time result mainly from the time required to read input data from files and the time spend for parallel communication. Overall, the experiments show that the computing time increases only by a small amount even in the case of an increase from 512 to 4096 processor cores (each executing one process of the parallel program). The increase in computing time is mainly due to the increased amount of data that is transferred between the single model task that computes the analysis step and all other model tasks, but also the reading of input files will influence the computation time. This computing time increase is, of course, specific to the computer that is used to conduct the experiments. In particular, the performances of its network and disk storage system will influence the execution times.

While the experiments used an assimilation system with online-coupling, a similar speedup can also be expected offline coupling. However, if the experiments of this section would be performed with a data assimilation system using offline coupling of model and analysis step, a moderate time overhead can be expected. Based on the model start-up time and the time to read and write ensemble states, a minimal time overhead of about 1.5% can be estimated for the smallest ensemble of 8 members and using 8 processes per model task. The maximal time overhead is reached for the large ensemble of 64 members and 64 processes per model task. It can be estimated to reach about 15%. In both cases, the additional required time is dominated by the start-up time of the model.

5.2. Speedup of the Analysis Step

While the computing time for the ensemble forecasts will dominate for many applications, there can be cases where the analysis step requires a significant amount of time. For example, this situation can occur if observations are very frequently assimilated. To examine the parallel performance of the analysis step, data assimilation experiments are conducted where the analysis step is performed after each time step. A single model task is used to integrate all ensemble members. The analysis step requires now up to 50% of the overall computing time. Using an offline-coupled assimilation system for this configuration would result strongly increased time requirements. The time overhead caused by multiple model restarts as well as writing and reading files holding ensemble states, would amount to about 60% for the smallest case of 8 processes. For the largest case using 320 processes, the required time can be estimated to be about 14 times larger than for the online-coupled assimilation system.

The left panel of Fig. 3 shows the speedup of the forecast phase and the analysis step between 8 and 320 processes. The speedup of the ensemble forecasts is again governed by the speedup of the model itself. For 320 processes the speedup reaches about the half of the ideal speedup, namely a parallel efficiency of 48%, but the efficiency is higher for smaller numbers of processors as was mentioned before. The speedup for the filter analysis is closer to the ideal speedup reaching a value of about 32 for the 40-fold increase from 8 to 320 processes. Overall, the parallel efficiency of the analysis step is close to 90% for up to 224 processes. For 320 processes a parallel efficiency of 80% is obtained. The efficiency is irregularly low for 256 processes. This effect will be explained below.

To assess the reasons for the somewhat irregular speedup behavior of the analysis step, the speedup of different parts of the analysis algorithm has to be examined. The speedup of the parts of the analysis step that were discussed in section 2.3 is shown in the right panel of Fig. 3. An ideal speedup is visible for most parts of the filter algorithm. These parts are the search for the observations that define a local observation domain, the local analysis update of the state estimate, and the local ensemble transformation. Also the initialization of the local state and ensemble and the reinitialization of the global quantities exhibit an ideal speedup. They are omitted from the figure for compactness as they require negligible time. In contrast, the non-local preparations of the analysis step do not exhibit any speedup. This part of the analysis step computes the innovation term term $\mathbf{y} - \mathbf{H}\overline{\mathbf{x}^f}$ and the observed ensemble $\mathbf{HL}$ on the full model grid[3]. It involves that each process reads the observations for its sub-domain from a file. Then, the global innovation and observed ensemble are gathered for each process. Because

[3]Using $\mathbf{y} - \mathbf{H}\overline{\mathbf{x}^f}$ and $\mathbf{HL}$ on the full model grid is actually a simplification of the implementation. Due to the localization, only the part of these quantities is required that resides within each individual observation sub-domain δ. As this data is distributed over several processes, their collection would require a very complicated communication structure, in particular for a model using an unstructured grid like FEOM. However, at least a moderate speedup of these operations could be expected with an implementation that only collects the necessary data.

of this, the amount of work performed by each process does not decrease with an increasing number of processes. Thus, no speedup can be expected. However, the non-local preparation part requires only a very small amount of the computing time. For 8 processes, it is about 0.6% of the overall time of the analysis step. As all other parts of the filter analysis show an ideal speedup, the relative time for the non-local preparations increases with the number of processes to about 21% for 320 processes. Accordingly, the non-local preparation part reduces the overall speedup. The execution time varies by a factor of two for different processor numbers. However, it does not show a systematic dependence on the number of processes. The variability in the execution time is caused by the file operations, which dominate the non-local preparations. This time was particularly high in case of 256 processes, which caused the particularly low speedup for the full filter analysis step visible in the left panel of Fig. 3. The observed speedup behavior is specific for the SEIK filter. However, the communication of information on $\mathbf{y}-\mathbf{H}\overline{\mathbf{x}^f}$ and $\mathbf{HL}$ between all processes is also required in other EnSKFs. Thus, the limitation of the speedup by these operations is a general property of EnSKFs.

The influence of the non-local preparation part of the analysis shows that it has to be implemented with care. As the operations are specific for the model and observations used, this part cannot be implemented in a generic and at the same time highly efficient way. Instead, one has to consider the specific properties of the observations as well as the model grid to obtain an efficient implementation. The easiest strategy could be to ensure that the execution time for this part is as low as possible as this will limit its influence on the total computing of the analysis step.

The speedup of the assimilation system depends on the load balancing. In the case discussed here observations were available at each grid surface point. Further, the mesh is partitioned such that the number of surface grid points is approximately equal for each process. Due to this configuration, the nearly optimal speedup was achieved. With other types of observations, like precipitation in an atmospheric model or hydrographic data from ARGO floats (see, e.g. Deng et al., 2010), the availability of the observations could be non-uniform. If the distribution of the observations is strongly non-uniform and the computing time of the analysis step is significant, it might be worth for optimal load balancing to use in the analysis step a distribution of the model grid in which the available observations are uniformly distributed. In this case, the grid distribution will be distinct in the forecast phase and the analysis step and the model fields have to be redistributed by parallel communication.

6. Conclusions

This study discussed implementation strategies for ensemble-based data assimilation algorithms with large-scale numerical models. In particular, the aspect of the parallelization of the full assimilation system consisting of the ensemble forecast and the analysis step was discussed. The analysis step of the filter algorithms can be efficiently implemented using a decomposition of the state domain. For the ensemble forecast, a domain decomposition for each single model instance can be combined with the possibility to let several models as model tasks compute the forecast of a different states at the same time.

For the implementation of the assimilation system the choice of offline and online implementations exists. In the offline implementation separate programs are used for the ensemble forecast and the analysis step. The information transfer between both programs is performed using files. Computationally more efficient is the online implementation where both the model and the analysis step are combined into a single program. In this case, the information transfer between the different parts is conduction using subroutine calls and direct memory access. It was shown that the combination of the model with the data assimilation environment that controls the ensemble forecasts and the analysis step can be performed by extending the model source code with a four subroutine calls plus an ensemble loop that encloses the time stepping part of the model. In particular, this implementation strategy does not require that the model itself is available as a subroutine.

Numerical experiments using an implementation of the finite element ocean model (FEOM) with the parallel data assimilation framework (PDAF) showed that the resulting data assimilation system provides a very good parallel performance. Even with a rather small configuration of the model, the program was able to efficiently use about 4000 processors. The overall scalability of the assimilation program thus allows to compute large-scale assimilation applications within a limited time span.

The PDAF software used here provides a generic implementation of the data assimilation environment following the implementation strategy discussed in this study. In addition, PDAF includes implementations of different parallelized ensemble-based Kalman filters. In the future it is planned to apply the framework for a wider range of applications and to use it as a general toolset for the development and study new filter algorithms.

In the light of new processor architectures, like GPU-based accelerators, and increasing numbers of processor cores and multiple levels of memory hierarchy, new parallelization options become of interest. Future work will examine the adaption of the analysis step in PDAF to hybrid parallelization using a combination of OpenMP and MPI as well as the use of GPUs.

Acknowledgements

The computations have been performed on the compute systems of the North-German Supercomputing Alliance (HLRN). The authors thank Dr. Ashwanth Srinivasan and a second anonymous reviewer for their helpful comments.

References

Anderson, J. L., 2001. An Ensemble Adjustment Kalman Filter for data assimilation. Monthly Weather Review 129, 2884–2903.

Bishop, C. H., Etherton, B. J., Majumdar, S. J., 2001. Adaptive sampling with the Ensemble Transform Kalman Filter. Part I: Theoretical aspects. Monthly Weather Review 129, 420–436.

Burgers, G., van Leeuwen, P. J., Evensen, G., 1998. On the analysis scheme in the Ensemble Kalman Filter. Monthly Weather Review 126, 1719–1724.

Ciavatta, S., Torres, R., Suax-Picart, S., Allen, J. I., 2011. Can ocean color assimilation improve biogeochemical hindcasts in shelf seas? Journal of Geophysical Research 116, C12043.

Danilov, S., Kivman, G., Schröter, J., 2004. A finite-element ocean model: Principles and evaluation. Ocean Modeling 6, 125–150.

Danilov, S., Schröter, J., 2010. Unstructured meshes in large-scale ocean modeling. In: Freeden, W., Zuhair, Z. M. (Eds.), Handbook of Geomathematics. Springer, pp. 371–398.

Deng, Z., Tang, Y., Wang, G., 2010. Assimilation of Argo temperature and salinity profiles using a bias-aware localized EnKF system for the Pacific Ocean. Ocean Modeding 35, 187–205.

Evensen, G., 1994. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. Journal of Geophysical Research 99 (C5), 10143–10162.

Evensen, G., 2004. Sampling strategies and square root analysis schemes for the EnKF. Ocean Dynamics 54, 539–560.

Gaspari, G., Cohn, S. E., 1999. Construction of correlation functions in two and three dimensions. Quarterly Journal of the Royal Meteorological Society 125, 723–757.

Gropp, W., Lusk, E., Skjellum, A., 1994. Using MPI - Portable Parallel Programming with the Message-Passing Interface. The MIT Press, Cambridge.

Hoteit, I., Pham, D.-T., Blum, J., 2002. A simplified reduced order Kalman filtering and application to altimetric data assimilation in Tropical Pacific. Journal of Marine Systems 36, 101–127.

Houtekamer, P. L., Mitchell, H. L., 2001. A sequential ensemble Kalman filter for atmospheric data assimilation. Monthly Weather Review 129, 123–137.

Hunt, B. R., Kostelich, E. J., Szunyogh, I., 2007. Efficient data assimilation for spatiotemporal chaos: A local ensemble transform Kalman filter. Physica D 230, 112–126.

Janjić, T., Nerger, L., Albertella, A., Schröter, J., Skachko, S., 2011a. On domain localization in ensemble based Kalman filter algorithms. Monthly Weather Review 139, 2046–2060.

Janjić, T., Schröter, J., Albertella, A., Bosch, W., Rummel, R., Schwabe, J., Scheinert, M., 2011b. Assimilation of geodetic dynamic ocean topography using ensemble based Kalman filter. Journal of Geodynamics in press, doi:10.1016/j.jog.2011.07.001.

Kalman, R. E., 1960. A new approach to linear filtering and prediction problems. Transactions of the ASME, Journal of Basic Engineering 82, 35–45.

Karypis, G., Kumar, V., 1999. A fast and high quality multilevel scheme for partitioning irregular graphs. SIAM Journal on Scientific Computing 20, 359–392.

Keppenne, C. L., 2000. Data assimilation into a primitive-equation model with a parallel ensemble Kalman filter. Monthly Weather Review 128, 1971–1981.

Keppenne, C. L., Rienecker, M. M., 2002. Initial testing of a massively parallel ensemble Kalman filter with the Poseidon isopycnal ocean circulation model. Monthly Weather Review 130, 2951–2965.

Nerger, L., 2003. Parallel filter algorithms for data assimilation in oceanography. Ph.D. thesis, University of Bremen, Germany.

Nerger, L., Danilov, S., Hiller, W., Schröter, J., 2006. Using sea level data to constrain a finite-element primitive-equation ocean model with a local SEIK filter. Ocean Dynamics 56, 634–649.

Nerger, L., Gregg, W. W., 2007. Assimilation of SeaWiFS data into a global ocean-biogeochemical model using a local SEIK filter. Journal of Marine Systems 68, 237–254.

Nerger, L., Hiller, W., Schröter, J., 2005a. A comparison of error subspace Kalman filters. Tellus 57A, 715–735.

Nerger, L., Hiller, W., Schröter, J., 2005b. PDAF - the Parallel Data Assimilation Framework: Experiences with Kalman filtering. In: Zwieflhofer, W., Mozdzynski, G. (Eds.), Use of High Performance Computing in Meteorology - Proceedings of the 11. ECMWF Workshop. World Scientific, pp. 63–83.

Nerger, L., Janjić, T., Schröter, J., Hiller, W., 2011. A unification of ensemble square-root filters. Monthly Weather Review, in press, doi:10.1175/MWR–D–11–00102.1.

Pham, D. T., 2001. Stochastic methods for sequential data assimilation in strongly nonlinear systems. Monthly Weather Review 129, 1194–1207.

Pham, D. T., Verron, J., Gourdeau, L., 1998a. Singular evolutive Kalman filters for data assimilation in oceanography. Comptes Rendus de l'Académie die Science Paris, Series II 326 (4), 255–260.

Pham, D. T., Verron, J., Roubaud, M. C., 1998b. A singular evolutive extended Kalman filter for data assimilation in oceanography. Journal of Marine Systems 16, 323–340.

Rollenhagen, K., Timmermann, R., Janjić, T., Schröter, J., Danilov, S., 2009. Assimilation of sea ice motion in a finite-element sea-ice model. Journal of Geophysical Research 114, C05007.

Skachko, S., Danilov, S., Janjić, T., Schröter, J., Sidorenko, D., Savcenko, R., Bosch, W., 2008. Sequential assimilation of multi-mission dynamical topography into a global finite-element ocean model. Ocean Science 4, 307–318.

Tippett, M. K., Anderson, J. L., Bishop, C. H., Hamill, T. M., Whitaker, J. S., 2003. Ensemble square root filters. Monthly Weather Review 131, 1485–1490.

van Leeuwen, P. J., 2009. Particle filtering in geophysical systems. Monthly Weather Review 137, 4089–4114.

Wang, Q., Danilov, S., Schröter, J., 2008. Finite element ocean circulation model based on triangular prismatic elements with application in studying the effect of topography representation. Journal of Geophysical Research 113, C05015.

Whitaker, J. S., Hamill, T. M., 2002. Ensemble data assimilation without perturbed observations. Monthly Weather Review 130, 1913–1927.

List of Figures

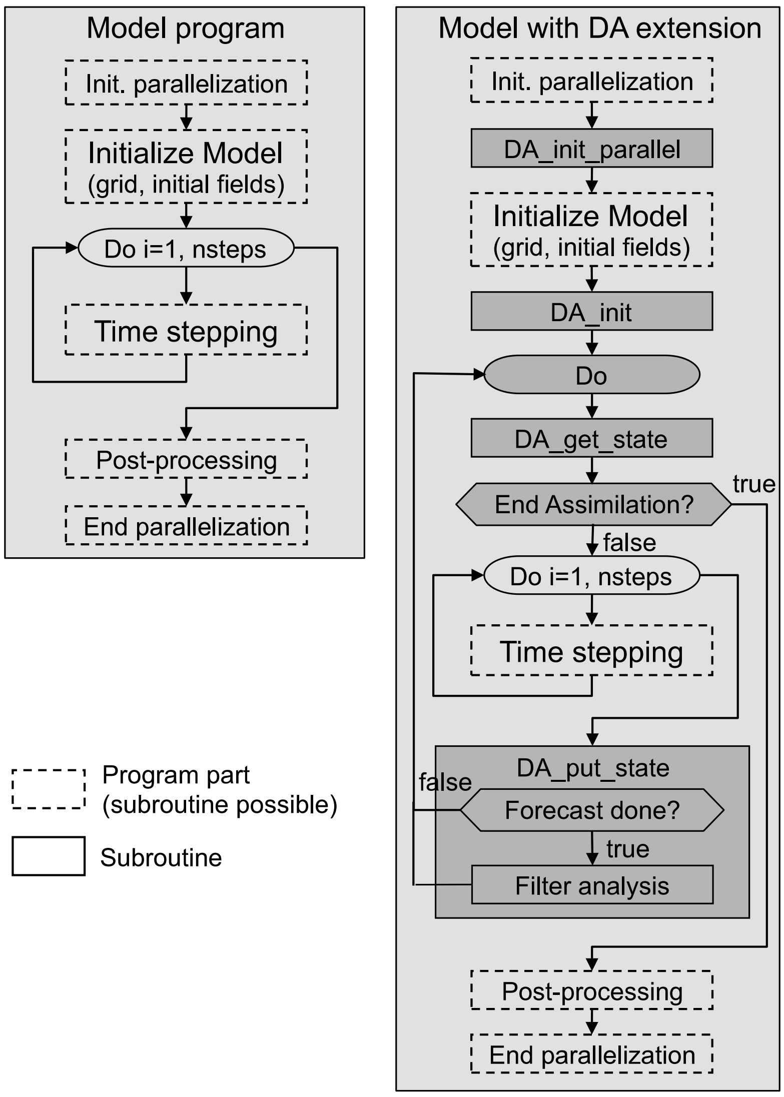
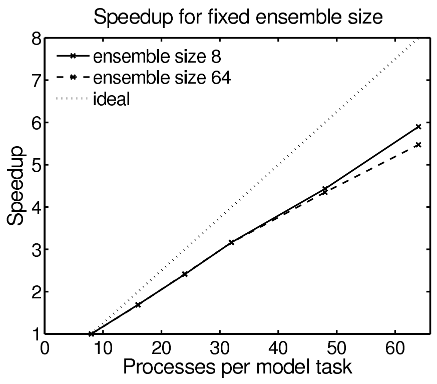


Figure 1: Left: Flow diagram of a typical numerical model; Right: Flow diagram of the model extended to an assimilation system by calls to routines of the assimilation framework. (Based on Nerger et al. (2005b))



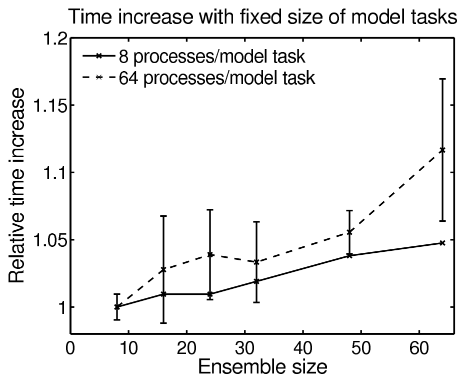
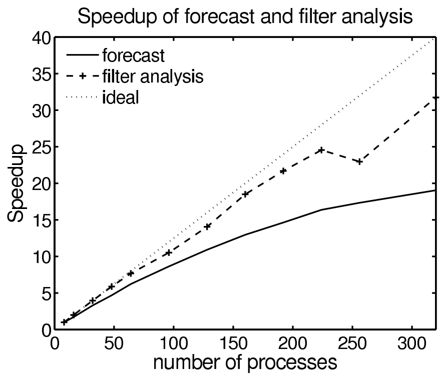


Figure 2: Parallel performance for an application in which the ensemble forecasts dominate the computing time. Left: Speedup of the ensemble data assimilation program for two fixed ensemble sizes. Right: Time increase for the case that the ensemble size is increased with a fixed size of each model task and each task integrating a single ensemble member. Error bars of one standard deviation of the execution time are shown for the case of 8 processes per model task.



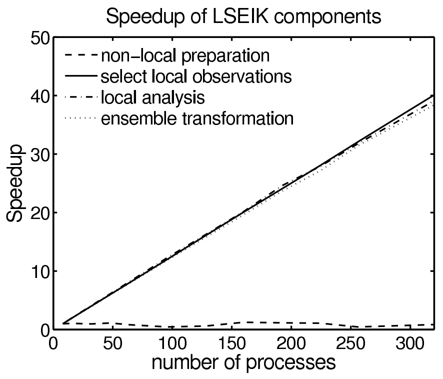


Figure 3: Parallel performance for an application in which the filter analysis step takes up to 50% of the total computing time. A single model task is used. Left: Speedup of the principal parts of the assimilation system. Right: Speedup of the components of the filter analysis step. The nonlocal data preparations limit the overall speedup.