PDAF - THE PARALLEL DATA ASSIMILATION FRAMEWORK: EXPERIENCES WITH KALMAN FILTERING

L. NERGER, W. HILLER AND J. SCHRÖTER

Alfred Wegener Institute for Polar and Marine Research,
P.O. Box 12 0161,
D-27515 Bremerhaven, Germany
E-mail: lnerger@awi-bremerhaven.de

in W. Zwieflhofer, G. Mozdzynski (Eds.), "Use of High Performance Computing in Meteorology", World Scientific, 2005, pp. 63-83

Data assimilation with advanced algorithms based on the Kalman filter and large-scale numerical models is computationally extremely demanding. This motivates the parallelization of the assimilation problem. As another issue, the implementation of a data assimilation system on the basis of existing numerical models is complicated by the fact that these models are typically not prepared to be used with data assimilation algorithms. To facilitate the implementation of parallel data assimilation systems, the parallel data assimilation framework PDAF has been developed. PDAF allows to combine an existing numerical model with data assimilation algorithms, like statistical filters, with minimal changes to the model code. Furthermore, PDAF supports the efficient use of parallel computers by creating a parallel data assimilation system. Here the structure and abilities of PDAF are discussed. In addition, the application of filter algorithms based on the Kalman filter is discussed. Data assimilation experiments show an excellent parallel performance of PDAF.

1. Introduction

Advanced data assimilation algorithms for filtering and smoothing applications with state-of-the-art large-scale geophysical models are of increasing interest. The applied algorithms are typically based on the Kalman filter[9]. Their aim is to estimate the state of a geophysical system (atmosphere, ocean, ...) on the basis of a numerical model and measurements by combining both sources of information. In addition, the algorithms provide an estimate of the error in the computed state estimate. There is already a large variety of data assimilation (DA) algorithms based on the Kalman

filter. Several of them can be classified as Error Subspace Kalman Filters (ESKF)[12], due to their representation of the estimation error in a low-dimensional sub-space of the true error space. Examples of such algorithms are the Ensemble Kalman Filter (EnKF)[6], the SEEK filter[15], and the SEIK filter[14]. These algorithms show several advantages in comparison to the variational DA algorithms 3D-Var and 4D-Var[16], which are widely used, e.g., in weather-forecasting applications. The variational algorithms do not provide a direct error estimate which, in contrast, is inherent to the ESKF algorithms. In addition, the 3D/4D-Var techniques require the implementation of an adjoint model code[5], which can be a very difficult task for realistic numerical models of the atmosphere and/or ocean. From the computational point of view, another advantage of the ESKF algorithms is their high scalability on parallel computers. While the 3D/4D-Var algorithms are of serial nature due to alternating integrations of the numerical (forward) model and its adjoint, the integration of an ensemble of model states in the ESKFs renders these algorithms to be highly scalable. Due to the differences between ESKF algorithms and the variational 3D/4D-Var methods, the ESKFs have the potential to provide better state estimates combined with error estimates in a smaller amount of time when they are applied in a highly parallel manner. This can significantly reduce the required computing time since the computational cost of advanced DA algorithms is several times higher than the cost of pure forecasts.

The implementation of DA systems on the basis of filter algorithms is typically complicated by the fact that the numerical models have not been developed with data assimilation applications in mind. This is partly also true for the existing DA systems which use variational techniques, since their code structure does often not allow a simple transformation into a filtering system. For the utilization of the parallelism in the ensemble integration the parallelization of the models has to be changed in a way to allow multiple concurrent model tasks. In addition, the filter algorithm itself has to be parallelized, to exploit the high scalability of the filter algorithms.

To facilitate the implementation of DA applications, the interface systems SESAM[17] and PALM[13] have been developed. These systems base on a logical separation between the filter and model parts of the DA problems. An interface structure for the transfer of data between the filter and the model is defined and the filter part is implemented such that it is independent of the model code itself. SESAM uses Unix shell scripts which control the execution of separated program executables like the numerical model and the program which computes, e.g., the assimilation of the measurement

in the analysis phase of the filter algorithm. Data transfers between the programs are performed using disk files. The structure of SESAM allows to use the model for DA without changes in the source code. The problem of data transfers between the model and filter parts of the data assimilation system is shifted to the issue of consistent handling of disk files. Since SESAM is based on shell scripts, it does not support multiple model tasks.

PALM uses program subroutines which have to be instrumented with meta information for the PALM system. The DA program is assembled using the prepared subroutines and a library of driver and algebraic routines supplied by PALM. The driver supports the concurrent integration of multiple model tasks. For the implementation of a DA application, PALM requires to assemble the algorithm from separate subroutines. Hence, also the model itself has to be implemented as a subroutine, since the main program is provided by the PALM-driver. In addition, the control of the filtering program will emanate from the driver routine of PALM.

With the introduction of the Parallel Data Assimilation framework PDAF we follow a different approach: For the creation of a data assimilation system, filter algorithms are attached to the model with minimal changes of the model source code itself. The parts of the filter problem which are model-dependent or refer to the measurements which are assimilated are organized as separate subroutines. These have to be implemented by the user of the framework. The data assimilation system is controlled by the user-written routines. Thus, the driver functionality remains in the model part of the program. Data transfers between the core part of PDAF and the user-supplied routines are conducted solely via a defined interface. Accordingly, the user-written routines can be implemented analogously to the model code, i.e. by sharing Fortran common blocks or modules. This simplifies the implementation of the user-supplied routines for users knowing about the particularities of their model. PDAF provides some of the most widely used filter algorithms, like the Ensemble Kalman Filter and the SEEK filter, which are fully implemented and optimized in the core part of PDAF. In general, also variational DA algorithms can be implemented within PDAF. However, in the current version of PDAF only filter algorithms are provided.

In the following, ESKF algorithms are reviewed in section 2 to discuss the structure and issues of these algorithms. Then, the PDAF system for the application of ESKF algorithms is presented in section 3. In section 4 experiments are discussed which show the parallel performance of PDAF and the EnKF, SEEK, and SEIK filter algorithms.

2. Error Subspace Kalman Filters

Error Subspace Kalman Filters are a class of advanced Kalman filter algorithms which are intended for data assimilation with large-scale nonlinear numerical models. Here, the concepts of ESKF algorithms are reviewed. A more detailed and mathematical description of three algorithms, the Ensemble Kalman filter, and the SEEK and SEIK filters together with a comprehensive review of the Kalman filter has been given by Nerger *et al.*[12].

The Kalman filter (KF)[9] is based on the theory of statistical estimation. Accordingly, the data assimilation problem is formulated as an estimation problem in terms of an estimate of the system state and a corresponding covariance matrix which describes the error in the state estimate. Since the error estimate is based on a covariance matrix, it is implicitly assumed that the errors are well described by Gaussian distributions. The virtue of the KF lies in the evolution of both the state estimate and the state covariance matrix according to the model dynamics. For linear dynamics, the KF can thus provide the optimal estimate if the system state is the error are normally distributed. The optimality is defined by the estimation errors which are of minimum variance and maximum likelihood. For nonlinear dynamics the extended Kalman filter (EKF), see Jazwinski[8], which is a first-order extension of the KF to nonlinear problems, can be applied. However, the estimate will be sub-optimal, since the assumption of Gaussian distributions is not conserved by nonlinear dynamics. Another practical issue represents the huge computational cost of the KF and EKF algorithms. Both require the explicit allocation of the state covariance matrix in memory. Furthermore, the evolution of the covariance matrix requires a number of model integrations which is twice as large as the dimension of the discretized model state. For today's models of the atmosphere or the ocean, the state dimension is of order 10^6 to 10^8. Thus, both the storage and the evolution of the covariance matrix are not possible with current high-performance computers.

To reduce the requirements of memory and computing time of the filter algorithm, several approximations and variants of the EKF have been developed over the last two decades in the atmospheric and oceanographic communities. A promising approach is given by the class of ESKF algorithms[12]. This class includes the popular ensemble Kalman filters (see Evensen[7] for a review), but also algorithms which use a low-rank approximation of the covariance matrix, like the SEEK filter[15]. In addition, the SEIK filter[14], which combines the ensemble and low-rank concepts, belongs to this class.

The concept of the ESKF algorithms is based on a low-rank approximation of the state covariance matrix. Mathematically, the ESKFs approximate the error space of the full EKF, which is described by the state covariance matrix, by a low-dimensional sub-space. The approximated covariance matrix is treated in decomposed form. This reduces the memory requirements. The evolution of the covariance matrix now only requires a number of model integrations which is equal to the rank of the approximated covariance matrix. Numbers as small as 7 have been reported for the rank in oceanographic applications[3]. Hence, the ESKF algorithms can strongly reduce the computation time for the DA in comparison to the full EKF. Furthermore, the ESKF algorithms allow for a better consideration of model nonlinearities. While the EKF applies a linearized model to evolve the covariance matrix, the Ensemble Kalman filter and the SEIK filter sample the covariance matrix together with the state estimate by an ensemble of model states which is integrated by the nonlinear model. The minimum size of the ensemble equals the rank plus one. The nonlinear ensemble integration can provide more realistic estimates of the state estimate and covariance matrix. In this case, the error sub-space estimated by the filter will no longer be a sub-space of the full error space estimated by the EKF, but a better estimate of the true error space of the estimation problem.

For the implementation of algorithms based on the Kalman filter, it is important to note that these methods are sequential filter algorithms, i.e. the algorithms can be separated into different phases which are executed sequentially. Figure 1 exemplifies the flow of a generic ESKF algorithm which relies on an ensemble representation of the covariance matrix. First, in the *initialization phase*, the initial state estimate and the corresponding covariance matrix are sampled by an ensemble of model states. Then, in the *forecast phase*, each ensemble state is evolved with the nonlinear numerical model. At times when measurements are available, the *analysis step* is performed: The analysis equations of the KF are applied to update the ensemble mean or all ensemble states together with the error estimate on the basis of the measurements and the evolved ensemble. Some algorithms, like the SEIK filter, apply finally a *re-initialization* step. Here the evolved ensemble of model states is transformed to represent the updated error estimate given by the decomposed covariance matrix. Subsequently, the algorithm steps back to perform the next forecast phase.

All phases of the filter algorithms can be parallelized. For the Ensemble Kalman filter this has been discussed in detail[10]. The parallelization of the SEEK and SEIK algorithms has also been studied and compared to that of

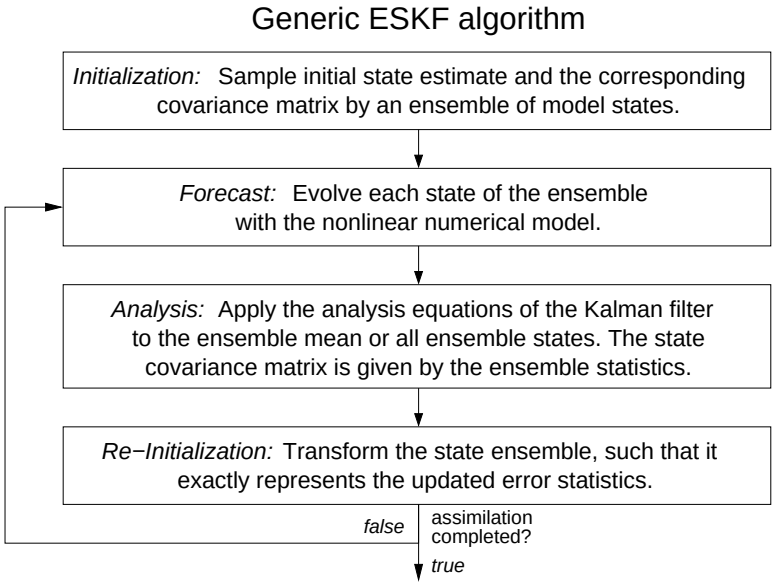

Figure 1. Generic flow diagram for error subspace Kalman filter algorithms using an ensemble of states.

the Ensemble Kalman filter[11]. There are basically two strategies which rely on the distribution of the matrix holding the ensemble states in its columns. First, it is possible to distribute the matrix column-wise over several processes. Thus, each process will hold a sub-ensemble of full model states. This strategy is independent from the model itself, as in the filter algorithm always full states are considered. The second strategy is to distribute the matrix row-wise. In this case, each processor will hold a full-ensemble of partial model states. This distribution can correspond to a decomposition of the model domain. It is thus the obvious parallelization strategy when the numerical model itself is parallelized using domain-decomposition. According to the distribution of the ensemble matrix, all other operations of the filter algorithm, in particular those in the analysis and re-initialization phases, are parallelized. The achievable parallel performance of the two parallelization strategies will be discussed in section 4.

3. The Parallel Data Assimilation Framework PDAF

The parallelization of ESKF algorithms alleviates the computational cost of storing and integrating an ensemble of model states and of the analysis

and re-initialization phases. However, the parallel implementation of an ESKF algorithm within an existing numerical model is a non-trivial task. The complex physical models are usually not prepared for the application of filter algorithms. Furthermore, while the core routines of filter algorithms can be parallelized independently from the model, the concurrent computation of multiple model integrations requires changes in the parallelization of the model itself.

To simplify the implementation of parallel DA systems, the Parallel Data Assimilation Framework PDAF has been developed. PDAF contains fully implemented and optimized ESKF algorithms and defines an application program interface (API) which permits to combine the filter algorithms with a numerical model. Only minimal changes in the model source code are required for the implementation. The core routines of PDAF are completely independent from the model and require no modifications during the implementation of a DA system. The API permits to switch easily between different filter algorithms or sets of measurements. Parts of the data assimilation program which are specific to the model or refer to measurements are held in separate subroutines. These have to be implemented by the user of the framework such that they can be called by the filter routines via the API. PDAF is implemented in Fortran95 using the MPI standard for parallelization. In general, and next to the distinction of mode-decomposed and domain-decomposed filter algorithms, two parallel configurations are possible for PDAF. First, the filter analysis and re-initialization phases can be executed by processes which also contribute to the model integrations. Second, processes which are separate from the processes computing the model integrations could perform the filter analysis and re-initialization phases. We will focus here on the first parallel configuration. However, the interface structure of PDAF is essentially equal for both configuration. The difference is mainly that for the first configuration, direct subroutine calls to the filter routines are possible while in the second configuration MPI communications will be necessary to connect the model integrations with the filter routines.

3.1. *General Considerations*

The development of PDAF, is based on the following considerations:

- The numerical model is independent from the routines of the filter algorithms. The model source code should be changed as little as possible when combining the filters with the model.

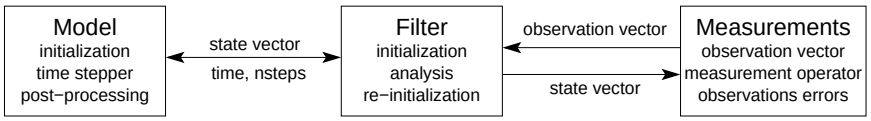

Figure 2. Logical parts of the data assimilation problem as considered in PDAF.

- The filter source code is independent from the model. It solely operates on model state vectors, not on the physical fields of the model.
- The information on measurements are independent from the filter and depend only on the information about the grid of the model. The model does not need information about the observations. To compute the filter analysis step the filter routines require information on the measurements. This is, e.g., the vector of measurements or the measurement operator, i.e. the operation which computes the observations which would be measured from the model state. Since this operation requires information on the spatial structure and the physical meaning of the elements of the state vector, it depends on the definition of the state vector and on the model grid. However, the filter algorithms only require the operation of the measurement operator on some state vector, which can be implemented as a subroutine.
- The framework can be logically partitioned into three parts as is sketched in figure 2. The transfer of information between the model and the filter as well as between the filter and the observations is performed via the API.
- PDAF has to allow for the execution of multiple concurrent model evolutions, each of these can be parallelized. Both, the parallelization of the model and the number of parallel model tasks have to be specified by the user.
- Like the model, the filter routines can be executed in parallel.
- The filter routines can either be executed by (some of) the processes used for the model integrations or by a set of processes which is disjoint from the set of model processes. The exact specification of processes has to be configured by the user.

To combine a filter algorithm with a numerical model in order to obtain a data assimilation program, we consider the 'typical' structure of a model

which computes the time evolution of several physical fields. The 'typical' structure is depicted in figure 3a. In the initialization phase of the program, the grid for the computations is generated and the physical fields are initialized. Subsequently, the model fields are integrated for *nsteps* time steps. The integration takes into account boundary conditions as well as external forcing fields, like wind fields over the ocean. Having completed the evolution some post-processing operations can be performed.

The following discussion of PDAF focuses on the configuration in which the filter routines are executed by some of the model processes. For this case, the structure of the DA program with attached filter is shown in figure 3b. To initialize the filter framework, a routine *pdaf_filter_init* is added to the initialization part of the program. Here the arrays required for the filter, like the ensemble matrix, are allocated. Subsequently, the state estimate and approximated covariance matrix are initialized and the state ensemble is generated in a user-supplied routine called by *pdaf_filter_init*. The major logical change when combining a filter algorithm with the model source code is that a sequence of independent evolutions has to be computed. This can be achieved by enclosing the time stepping loop by an unconditioned outer loop which is controlled by the filter algorithm. Before the time stepping loop of the model is entered, the subroutine *pdaf_get_state* is called. This routine provides a model state from the state ensemble together with the number of time steps (*nsteps*) to be computed. To enable the consistent application of time dependent forcing in the model, the *pdaf_get_state* also provides the current model time. In addition, a flag *doexit* is initialized which is used as an exit-condition for the unconditioned outer loop. For the forecast, the user has to assure that the model integrations are really independent. Any re-used fields must be newly initialized, for example. After the time stepping loop a call to the subroutine *pdaf_put_state* is inserted into the model source code. In this routine, the evolved model fields are stored back as a column of the ensemble state matrix of the filter. If the ensemble forecast has not yet finished, no further operations are performed in the routine *pdaf_put_state*. If, however, all model states of the current forecast phase are evolved, *pdaf_put_state* executes the analysis and re-initialization phases of the chosen filter algorithm. With this structure of PDAF, the logic to perform the ensemble integration is contained in the routines *pdaf_get_state* and *pdaf_put_state*. Furthermore, the full filter algorithms are hidden within these routines. The user is required to ensure independent model integrations. In addition, some routines have to be supplied by the user which provide operations related to

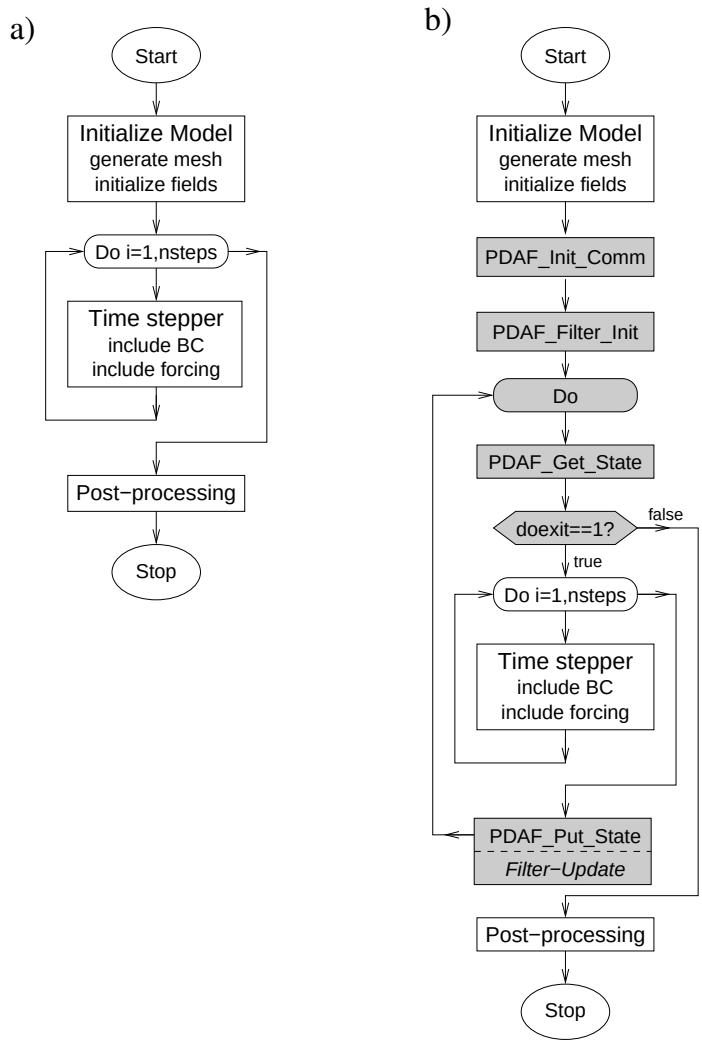

Figure 3. Flow diagrams: a) Sketch of the typical structure of a model performing time evolution of some physical fields. b) Structure of the data assimilation configuration of the model with attached filter. Added subroutine calls and control structures are shaded in gray.

the model fields or the measurements, as will be discussed below.

For the parallelization of the DA system, the routine *pdaf_init_comm* is added to the program. Details of this routine are discussed in section 3.3.

3.2. *Interface Structure*

The three subroutines *pdaf_filter_init*, *pdaf_get_state*, and *pdaf_put_state* provide a defined interface to the filter algorithms. In addition, the user-supplied routines like the observation-related subroutines are called using the API defined by PDAF. This structure allows to keep the core part of PDAF independent from the model, such that PDAF can be compiled independently from the model code. Thus, the API allows to use PDAF with models implemented in Fortran as well as in C. The user-supplied routines should be programmed similarly to the model code. This simplifies the inclusion of model-related variables, like information on the grid or the model fields.

For the case that the filter routines are executed by processes which compute also model integrations, the filter algorithms are split into parts which are contained in the three routines *pdaf_filter_init*, *pdaf_get_state*, and *pdaf_put_state*. However, all variables which are required to specify the filter are contained in the interface to *pdaf_filter_init*. Here also the parallelization information, like communicators, is handed over to PDAF. The three routines also require the specification of the names of user-supplied routines in the API. This allows the user to choose arbitrary names for these routines, since the internal calls are independent from the real subroutine names. Furthermore, it allows to switch easily, e.g., between different observational data sets. The specifications for each particular set of measurements can be contained in separate routines.

Next to the user-supplied routines required for the analysis phase of a filter algorithm, a pre/poststep routine has to be supplied. This routine is called at the begin of the DA problem, before each analysis phase, and after the re-initialization phase. This routine allows the user to conduct a further examination of the estimates of state and covariance matrix. Also corrections to the model state can be performed, which might be necessary after the statistical update of states by the filter algorithm[1].

The filter algorithms operate on abstract one-dimensional state vectors which are contained in the array of ensemble states. The transfer of state information between this ensemble matrix and the model fields is performed by other user-supplied routines. These are defined in the calls to the routines *pdaf_get_state* and *pdaf_put_state*. Before the integration of a state an operation *distribute_state* is required. This initializes the model fields from a state vector and performs other re-initializations of the model, if required. After the integration, the state vector is update from the evolved

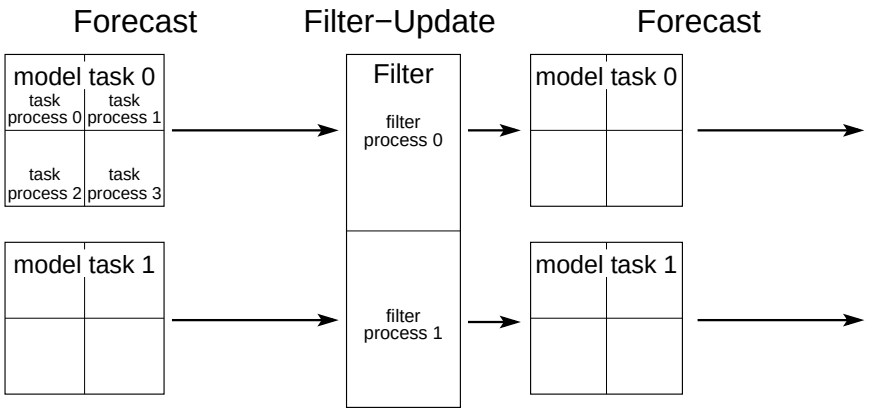

Figure 4. Two-level parallelism of PDAF: The model and filter parts of the program are parallelized. In addition, several model tasks can be executed concurrently.

model fields by an operation *collect_state*. Dependent on the parallelization of the model and on the choice whether a mode- or a domain-decomposed filter algorithm is used, some MPI communication can be necessary in these operations. A similar operation like *collect_state* is also required in the user-supplied routine which is called by *pdaf_filter_init* to initialize the ensemble matrix.

3.3. *Parallelization Aspects*

PDAF supports a 2-level parallelism, which is depicted in figure 4. Each model task and the filter routines themselves can be executed by multiple processes. In addition, multiple model tasks can be executed concurrently. The communication within the filter routines and the control of multiple model tasks is conducted within PDAF. Thus, only a possible communication for the initialization of the model fields before a model integration has to be added as described above. If the filter routines are executed by some of the processes which perform the model integrations, the four routines of PDAF which are called from the model code are always called by all processes. The decision which processes execute, e.g., the filter analysis phase is drawn within these routines.

For the parallelization of the DA system, a change to the model source code concerning the configuration of MPI communicators is required. For MPI-parallelized models there is typically a single model task which operates often in the global MPI communicator *MPI_COMM_WORLD*. To

allow for multiple concurrent model tasks, the global communicator has to be replaced by a communicator of disjoint process sets in which each of the model tasks operates. Thus, a communicator *COMM_MODEL* has to be defined. In the model source code, the reference to *MPI_COMM_WORLD* has to be replaced by *COMM_MODEL*. Besides the communicator for the model tasks, a communicator *COMM_FILTER* has to be defined for the processes which execute the filter routines. Finally, a communicator *COMM_COUPLE* is required, which defines processes used for data transfers between the filter and model parts of the DA system. These communicators are defined in the routine *pdaf_init_comm* which is inserted into the original model code as depicted in figure 3b. Since the process configurations will depend on the parallelization of the model, PDAF can only provide a template for *pdaf_init_comm*. This template will be utilizable in general, but without providing the optimal parallel performance. Thus, for optimal performance, the routine should be adapted to the particular problem, e.g. to support particular process topologies.

The communication is determined by the three MPI communicators *COMM_MODEL*, *COMM_FILTER*, and *COMM_COUPLE*. Figure 5 exemplifies a possible configuration of the communicators for a mode-decomposed filter. In the example the program is executed by a total of 8 processes. These are distributed into four parallel model tasks each executed by two processes in the context of *COMM_MODEL*. The filter routines are executed by two processes. These are the processes of rank 0 and 4 in the context of *MPI_COMM_WORLD*. In the context of *COMM_MODEL* the filter processes have rank 0. Each filter process is coupled to two model tasks. Thus, there are two disjoint process sets in *COMM_COUPLE* each consisting of two processes. With this configuration, the filter initialization in *pdaf_filter_init* divides the ensemble matrix into two matrices of sub-ensembles which are stored on the two filter processes. To utilize of all four model tasks, each filter process will again distribute its sub-ensemble at the begin of each forecast phase to the two model tasks which are coupled to it by *COMM_COUPLE*. After the forecast phase, the ensemble members are gathered again on the two filter processes. A simpler parallelization variant would be to execute the filter routines by a single process of each model task. This variant would also involve a smaller amount of communication.

If a domain-decomposition is used for the parallelization of the model and the filter parts of the program, a different configuration of the processes is used. Considered is again the situation that the filter and the model use the same domain-decomposition. Figure 6 shows a possible process

⟶ logical process number (= rank in *MPI_COMM_WORLD*)								
[0	1	2	3	4	5	6	7]	*MPI_COMM_WORLD*
[0	1]	[0	1]	[0	1]	[0	1]	*COMM_MODEL*
[0		1]		[0		1]		*COMM_COUPLE*
[0				1]				*COMM_FILTER*

Figure 5: Example communicator configuration for the case that the filter is executed by some of the model processes and the filter routines use a mode-decomposition o the ensemble matrix.

⟶ logical process number (= rank in *MPI_COMM_WORLD*)						
[0	1	2	3	4	5]	*MPI_COMM_WORLD*
[0	1	2]	[0	1	2]	*COMM_MODEL*
[0			1]			*COMM_COUPLE*
	[0			1]		*COMM_COUPLE*
		[0			1]	*COMM_COUPLE*
[0	1	2]				*COMM_FILTER*

Figure 6: Example communicator configuration for the case of domain-decomposed states. The filter is executed by some of the model processes.

configuration. Here the program is executed by six processes in total. These are distributed over two model tasks each consisting of three processes. The filter routines are executed by all processes of one of the model tasks. This allows for the direct initialization of the model fields in this task. The second model task is connected to the filter via *COMM_COUPLE*. With domain-decomposition, the initialization of the sub-states is performed in the user-supplied initialization routine called by *pdaf_filter_init*. The filter operates on the whole ensemble of local sub-states. To use multiple model tasks the ensemble is distributed into sub-ensembles. These are sent to the model tasks via *COMM_COUPLE*. The simplest process configuration with domain-decomposition would be to use a single model task only. If the filter and the model use the same domain-decomposition, this configuration would avoid any communication between the filter and the model.

3.4. *Filter algorithms*

In the current version PDAF contains implementations of the Ensemble Kalman Filter (EnKF)[6], the SEIK filter[14], and the SEEK filter[15]. Also some variants of the filters are available, e.g. a fixed-basis SEEK filter. This

formulation keeps the error directions, described by the singular vectors of the state covariance matrix, constant while changing the weight for these directions during the analysis phase of the filter algorithm. The fixed-basis SEEK filter allows for a much faster assimilation process, since only a single model integration is required. However, this variant is expected to provide inferior state and error estimates, see e.g. Brasseur *et al.*[2].

The clear separation between the model, measurement, and filter parts of PDAF facilitates the development and implementation of additional filter algorithms within PDAF. The filter implementation is independent from a further development of the model. Each implemented filter algorithm can be called from the model code via the uniform API. If a new filter algorithm requires new measurement-related routines, their names can be added to the call to *pdaf_put_state* to maintain the flexibility of user-defined subroutine names. To avoid that the API of *pdaf_put_state* becomes too long, separate routines for the filters have been implemented. These filter-specific routines only require the specification of the routines used in the particular algorithm.

4. Parallel Performance of PDAF and ESKF Algorithms

To exemplify the parallel performance of PDAF, the framework was tested in the finite element ocean model FEOM[4]. To allow for a large number of experiments, a small model configuration of a rectangular box of 31 by 31 grid points with 11 layers was used. In this model configuration synthetic observations of the full sea surface height field are assimilated each 2.5 days for a period of 40 days. Using the horizontal velocity components, the temperature field, and the sea surface height as state variables, the state vector has a size of 32674. In each analysis phase, a measurement vector of size 961 is assimilated. The experiments have been performed on a Sun Fire 6800 with 24 processors. Details on these experiments and a comparison of the filtering performances of different filter algorithms have been given by Nerger *et al.*[11].

Figure 7 shows the speedup of the total computing time as the function of the number of parallel model tasks. Results for the EnKF, SEEK, and SEIK filters are shown for an ensemble of 36 members. Each model task is executed by a single process. Thus, no MPI communication within the operations *distribute_state* and *collect_state* is required. The speedup is equal for all three filters and slightly sub-optimal with a parallel efficiency of about 85% for 18 model tasks. This sub-optimality is due to the

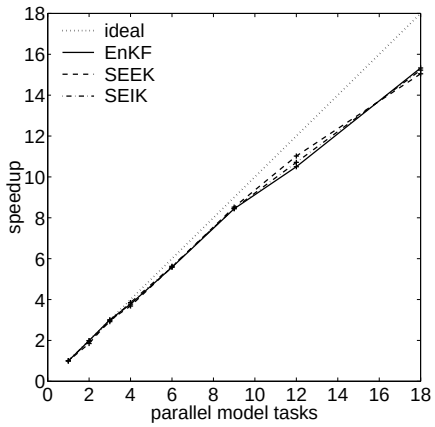

Figure 7. Speedup as a function of the number of parallel model tasks for the framework with a filter process on each model task. For 18 parallel model tasks a parallel efficiency of 85% is obtained.

iterative solver applied in the inverse time stepping of FEOM. Since the number of iterations can vary in each time step for different model tasks, a small de-synchronization occurs causing the speedup to be sub-optimal. The computing time for the filter analysis and re-initialization phases are negligible in these experiments. For the EnKF, which is the most costly algorithm in the experiments, the filter analysis requires less than 0.1% of the total computing time in the serial experiment.

There are practical situations in which the computing time for the filter analysis and re-initialization phases can become significant, e.g., if measurements are assimilated very frequently. In this case, the parallel efficiency of these parts of the algorithms will determine the speedup of the DA program. Figure 8 shows the computing time and speedup of the EnKF, SEEK, and SEIK algorithms for PDAF with mode-decomposition (upper panels) and domain-decomposition (lower panels) for ensembles with 60 members. For each filter algorithm and number of processes 20 experiments are performed. The figure shows mean values and error bars of one standard deviation. The SEEK and SEIK algorithms are much faster than the EnKF algorithm. This difference will diminish for increasing ensemble sizes, showing the difference of the algorithmic formulation of the EnKF compared to the SEEK and SEIK filters.

The speedup of the EnKF stagnates at a value of about 1.2 for both the mode-decomposition and domain-decomposition variants. For increasing

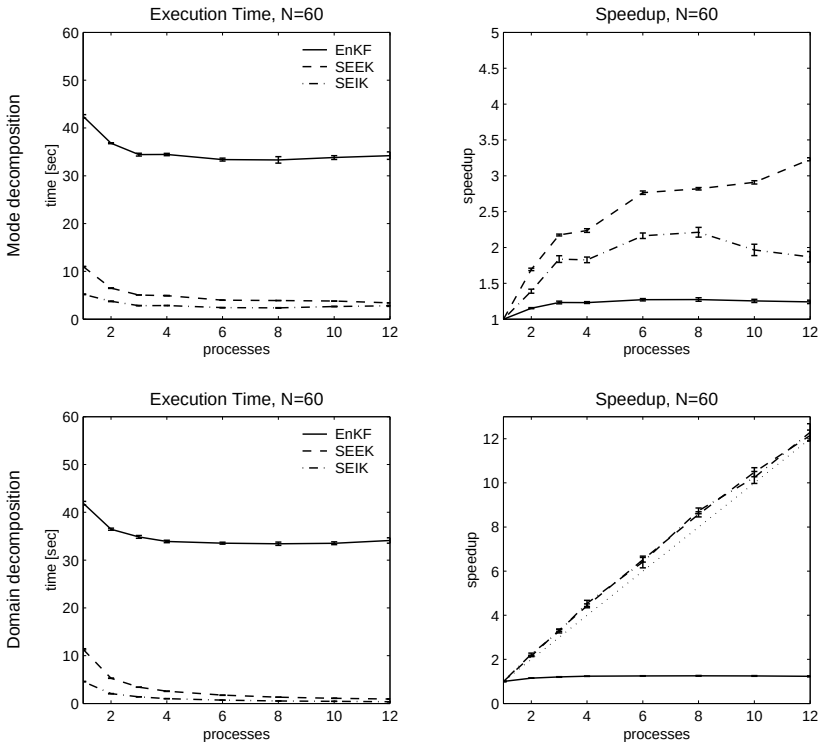

Figure 8. Execution time and speedup for the filter update phases (analysis and re-initialization) as a function of the number of processes. The assimilations have been performed with an ensemble size of 60. The upper panels show the results for the mode-decomposed variant of PDAF while the lower panels are for the domain-decomposition variant. Displayed are mean values and standard deviations over 20 experiments for each combination of filter algorithm and number of processes. The mode-decomposed algorithms require a large amount of communication which leads to a small speedup. For domain-decomposition an ideal speedup is obtained with the SEEK and SEIK filters.

ensemble sizes, the speedup will grow (not shown), but will remain far from optimal. This small speedup is particular to the EnKF algorithm and the possibilities of its parallelization. The operations in the EnKF algorithm are global over the ensemble and over the observation space. Due to this, a large amount of communication is required with mode-decomposition as well as domain-decomposition. In addition, several parts of the algorithm show a small parallel efficiency which could only be avoided by rearranging larger arrays over all processes which would itself require costly communications. A possibility to increase the speedup of the EnKF algorithm would

be to introduce a localization to the analysis phase, see e.g. Keppenne and Rienecker[10]. This would effectively amount to a reduction of the dimension of the measurement vector and hence to a smaller amount of MPI communication. However, a localization would involve an approximation of the analysis phase.

The speedup of the SEEK and SEIK filters depends strongly on the chosen parallelization strategy. For mode-decomposition only a small speedup is obtained. It will increase slightly for larger ensemble sizes. The small speedup is mainly due to a large amount of communication required in the mode-decomposed algorithms. For increasing ensemble size, the time for communications decreases relative to the time of computations. The difference in the speedups of the SEEK and SEIK algorithms is caused by algorithmic differences which lead to smaller communication requirements in the SEEK filter compared to the SEIK filter. For domain-decomposition the SEEK and SEIK filters show an ideal speedup. In this case only a very small amount of data is communicated. This is due to the fact, that both the SEEK and SEIK algorithms are global over the ensemble, i.e. over the error space, but not global over the model space. Thus, it is optimal to distribute data as full ensembles of sub-states as is done in the domain-decomposition variant of the filter algorithms. For larger ensemble sizes, the parallel efficiency will decrease slightly. This effect is caused by some non-parallelized operations whose complexity scales with the cube of the ensemble size. However, the influence of these operations will be negligible in practical situations with a much larger state dimension than in the experiments performed here.

5. Summary

The parallel data assimilation framework PDAF has been introduced for data assimilation with advanced data assimilation algorithms based on the Kalman filter. To motivate the organization of the framework, the concept and structure of error subspace Kalman filters has been reviewed. These algorithms represent a wide range of filter algorithms with promising properties for their application to large-scale nonlinear numerical models. The filters show a sequential structure consisting of alternating forecast and analysis phases. In addition, some algorithms perform a re-initialization phase. The filter algorithms allow for a clear separation between the core part of a filter, the model used to integrate different model states, and the measurements to be assimilated during the analysis phase. In addition, the

algorithms share a natural parallelism in the forecast phase caused by the independent integration of an ensemble of model states.

These properties motivate the development of a data assimilation framework which facilitates the implementation of a data assimilation system on the basis of numerical models which are not designed for data assimilation. PDAF defines an application program interface used to call the framework routines from within the model code. In addition, an interface is defined for several user-supplied routines which, e.g., include operations dependent on the measurements. These routines are called by the core routines of PDAF. The framework contains a number of filter algorithms which are fully parallelized and optimized. In the current version of PDAF these are the popular Ensemble Kalman Filter, the SEEK filter, and the SEIK filter. Two parallelization variants are supported which rely on different distributions of the matrix of ensemble states. The variants are the column-wise mode-decomposition and the row-wise domain-decomposition. The domain-decomposition variant is the obvious strategy if the model itself is parallelized using a domain-decomposition of the model grid. The framework permits a further development and implementation of DA algorithms independently from the development of the numerical models. Switching between the filter algorithms is possible by the specification of a single variable. In general, also alternative methods like the variational 4D-Var (adjoint) techniques are possible within the framework, though this has not yet been implemented.

Numerical experiments with an idealized configuration of the finite element ocean model FEOM have been performed to examine the parallel efficiency of the full framework as well as the analysis and re-initialization parts of the filter algorithms. An excellent speedup is obtained for the full framework. With regard to the analysis and re-initialization phases of the filters, strong differences between the Ensemble Kalman filter and the SEEK and SEIK filters have been found. The EnKF shows only a small speedup for both decomposition variants. The SEEK and SEIK filters show a small speedup for mode-decomposition but exhibit an ideal speedup for the variant using domain-decomposition with small ensemble sizes.

It is planned to make PDAF publicly available in the future to facilitate the implementation of data assimilation systems. Furthermore, a simplified sharing of filter algorithms will be possible on this unified implementation basis. Currently the framework undergoes a beta-testing phase with a limited number of users and models.

Acknowledgments

We are grateful to Stephan Frickenhaus, Sergey Danilov and Gennady Kivman for instructive discussions and comments on the structure and parallelization of PDAF. Meike Best is thanked for a careful proofreading.

References

1. J.-M. Brankart, C.-E. Testut, P. Brasseur, and J. Verron. Implementation of a multivariate data assimilation scheme for isopycnic coordinate ocean models: Application to a 1993-1996 hindcast of the North Atlantic ocean circulation. *J. Geophys. Res.*, 108(C3):3074, 2003. doi:10.1029/2001JC001198.
2. P. Brasseur, J. Ballabrera-Poy, and J. Verron. Assimilation of altimetric data in the mid-latitude oceans using the singular evolutive extended Kalman filter with an eddy-resolving, primitive equation model. *J. Mar. Syst.*, 22:269–294, 1999.
3. K. Brusdal, J. M. Brankart, G. Halberstadt, G. Evensen, P. Brasseur, P. J. van Leeuwen, E. Dombrowsky, and J. Verron. A demonstration of ensemble based assimilation methods with a layered OGCM from the perspective of operational ocean forecasting systems. *J. Mar. Syst.*, 40-41:253–289, 2003.
4. S. Danilov, G. Kivman, and J. Schröter. Evaluation of an eddy-permitting finite-element ocean model in the North Atlantic. *Ocean Modelling*, 2005. in press.
5. F.-X. Le Dimet and O. Talagrand. Variational algorithms for analysis and assimilation of meteorological observations: Theoretical aspects. *Tellus*, 38A:97–110, 1986.
6. G. Evensen. Sequential data assimilation with a nonlinear quasi-geostrophic model using Monte Carlo methods to forecast error statistics. *J. Geophys. Res.*, 99(C5):10143–10162, 1994.
7. G. Evensen. The Ensemble Kalman Filter: Theoretical formulation and practical implementation. *Ocean Dynamics*, 53:343–367, 2003.
8. A. H. Jazwinski. *Stochastic Processes and Filtering Theory*. Academic Press, New York, 1970.
9. R. E. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, J. Basic Eng.*, 82:35–45, 1960.
10. C. L. Keppenne and M. M. Rienecker. Initial testing of a massively parallel Ensemble Kalman Filter with the Poseidon isopycnal ocean circulation model. *Mon. Wea. Rev.*, 130:2951–2965, 2002.
11. L. Nerger. *Parallel Filter Algorithms for Data Assimilation in Oceanography*. Number 487 in Reports on Polar and Marine Research. Alfred Wegener Institute for Polar and Marine Reasearch, Bremerhaven, Germany, 2004. PhD Thesis, University of Bremen.
12. L. Nerger, W. Hiller, and J. Schröter. A comparison of error subspace Kalman filters. *Tellus A*, 2005. in press.
13. Project PALM. Projet d'assimilation par logiciel multi-methodes. URL `http://www.cerfacs.fr/~palm/`.

14. D. T. Pham, J. Verron, and L. Gourdeau. Singular evolutive Kalman filters for data assimilation in oceanography. *C. R. Acad. Sci., Ser. II*, 326(4):255–260, 1998.
15. D. T. Pham, Jacques Verron, and Marie Christine Roubaud. A singular evolutive extended Kalman filter for data assimilation in oceanography. *J. Mar. Syst.*, 16:323–340, 1998.
16. F. Rabier, H. Järvinen, E. Klinker, J.-F. Mahfouf, and A. Simmons. The ECMWF operational implementation of four-dimensional variational assimilation. 1: Experimental results with simplified physics. *Quart. J. Roy. Meteor. Soc.*, 126:1143–1170, 2000.
17. SESAM. An integrated system of sequential assimilation modules. URL `http://meol715.hmg.inpg.fr/Web/Assimilation/SESAM/`.