PDAF Tutorial

Implementation of the analysis step

in online mode with a parallel model

using PDAF's full interface

http://pdaf.awi.de

PDAF Parallel
Data Assimilation
Framework

Implementation Tutorial for PDAF online with serial model

We demonstrate the implementation

of an offline analysis step with PDAF

using the template routines provided by PDAF

The example code is part of the PDAF source code package
downloadable at http://pdaf.awi.de

(This tutorial is compatible with PDAF V3.0 and later)

Please note:

The implementation variant described here is rather for reference with
older implementations or can be considered as an expert-mode.
We recommend to base any new implementation on using the PDAF3
interface that was introduced in PDAF V3.0. Please see the tutorial.

PDAF Parallel
Data Assimilation
Framework

Implementation Tutorial for PDAF online / parallel model

This is just an example!

For the complete documentation of PDAF's interface

see the documentation

at http://pdaf.awi.de

PDAF Parallel
Data Assimilation
Framework

Overview

Focus on Error Subspace Transform Kalman Filter
     (ESTKF, Nerger et al., Mon. Wea. Rev. 2012)

4 Parts

  a) Global filter   b) Localized filter

We recommend to first implement the global filter. The localized
filter re-uses routines of the global filter.

In this tutorial we only cover the case of a parallel model.
The implementation using a model without parallelization is
described in a separate tutorial.

PDAF Parallel
Data Assimilation
Framework

Contents

PDAF Parallel
Data Assimilation
Framework

0a) Files for the Tutorial

PDAF Parallel
Data Assimilation
Framework

Tutorial implementation

Files are in the PDAF package

Directory:

`/tutorial/classical/online_2D_parallelmodel`

- Fully working implementations of user codes

- PDAF core files are in `/src`
  Makefile refers to it and compiles the PDAF library

- Only need to specify the compile settings (compiler, etc.) by environment variable PDAF_ARCH. Then compile with 'make'.

PDAF Parallel
Data Assimilation
Framework

Template files for online mode

Directory: `/templates/online`

- Contains all required files

- Contains also
  command line parser
  (convenient but not required)

To generate your own implementation:

1. Copy content of directory
   e.g. into sub-directory of model source code

2. Add calls to interface routines to model code

3. Complete user-routines for your model

4. Adapt compilation (e.g. Makefile) and compile

5. Run with assimilation options

PDAF Parallel
Data Assimilation
Framework

PDAF library

Directory: `/src`

- The PDAF library is not part of the template
- PDAF is compiled separately as a library
  and linked when the assimilation program is compiled
- Makefile includes a compile step for the PDAF library
- One can also cd to `/src` and run 'make' there
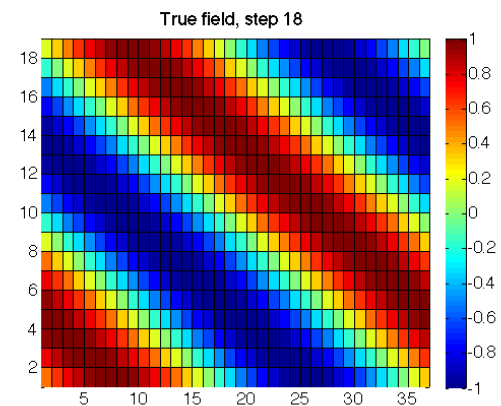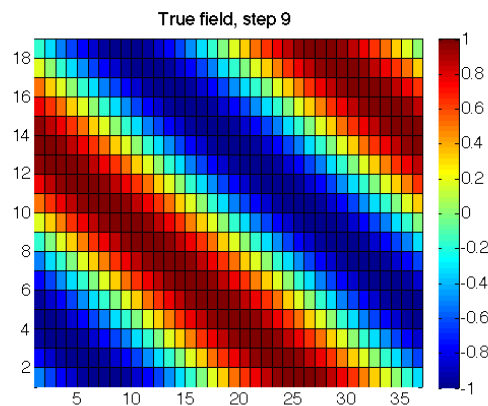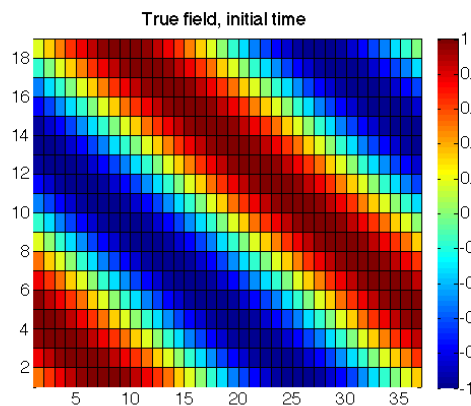 (requires setting of PDAF_ARCH)

`$PDAF_ARCH`

- Environment variable to specify the compile specifications
- Definition files in `/make.arch`
- Define by, e.g.
  `setenv PDAF_ARCH linux_gfortran` (tcsh/csh)
  `export PDAF_ARCH=linux_gfortran` (bash)

PDAF Parallel
Data Assimilation
Framework

0b) The parallelized model

2D „Model"

- See the separate tutorial slides about the model

- Simple 2-dimensional grid domain

- 36 x 18 grid points (longitude x latitude)

- True state: sine wave in diagonal direction
  (periodic for consistent time stepping)

True field, initial time

True field, step 9

True field, step 18

PDAF Parallel
Data Assimilation
Framework

Parallel `Model`: Files

The source code of the parallel model consists of the following files:

- mod_model.F90

- mod_parallel_model.F90

- main.F90

- initialize.F90

- integrate.F90

Note: One can nicely compare the source codes of the model without and with parallelization

For clarity, the implementation with PDAF is found in

- main_pdaf.F90

- integrate_pdaf.F90

It allows for easy comparison of the implementations

PDAFParallel
Data Assimilation
Framework

0c) state vector and observation vector

State vector – some terminology used later

- PDAF performs computations on state vectors

- **State vector**

  - Stores model fields in a single vector

  - Tutorial shows this for one 2-dimensional field

  - Multiple fields are just concatenated into the vector

  - All fields that should be modified by the assimilation have to be in the state vector

- **State dimension**

  - Is the length of the state vector
    (the sum of the sizes of the model fields in the vector)

- **Ensemble array**

  - Rank-2 array which stores state vectors in its columns

PDAF Parallel
Data Assimilation
Framework

Observation vector

- **Observation vector**

  - Stores all observations in a single vector

  - Tutorial shows this for one 2-dimensional field

  - Multiple observed fields are just concatenated into the vector

- **Observation dimension**

  - Is the length of the observation vector
    (sum of the observations over all observed fields in the vector)

- **Observation operator**

  - Operation that computes the observed part of a state vector

  - Tutorial only selects observed grid points

  - The operation can involve interpolation or integration
    depending on type of observation

PDAF Parallel
Data Assimilation
Framework

0c) PDAF online mode

Online mode

- Combine model with PDAF into single program

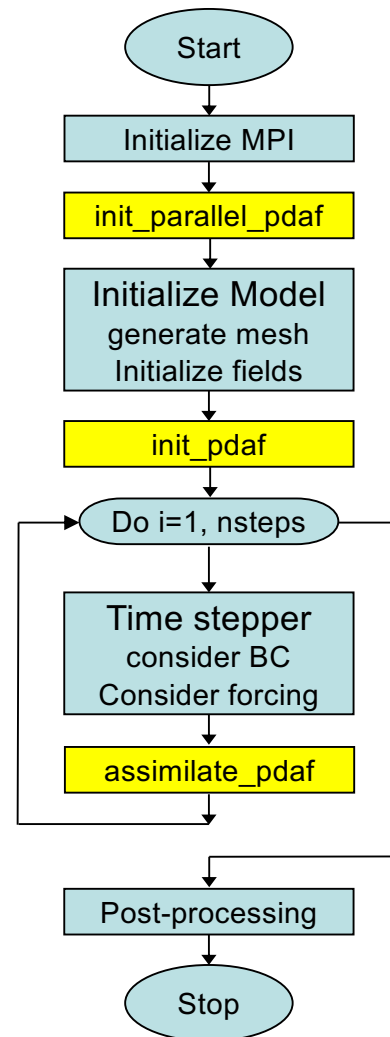 - "`model_pdaf`"

- Add 3 subroutine calls:

  `init_parallel_pdaf`   - revise parallelization

  `init_pdaf`   - initialize assimilation

  `assimilate_pdaf`   - perform assimilation

- Implement user-supplied routines, e.g. for

  - observation operator

  - initialization of observation vector

  - transfer between state vector and model fields

PDAF Parallel
Data Assimilation
Framework

Program flow with model extended for data assimilation

Simulation Model

Assimilation System

Start → Initialize MPI → Initialize Model generate mesh Initialize fields → Do i=1, nsteps → Time stepper consider BC Consider forcing → Post-processing → Stop

Start → Initialize MPI → init_parallel_pdaf → Initialize Model generate mesh Initialize fields → init_pdaf → Do i=1, nsteps → Time stepper consider BC Consider forcing → assimilate_pdaf → Post-processing → Stop
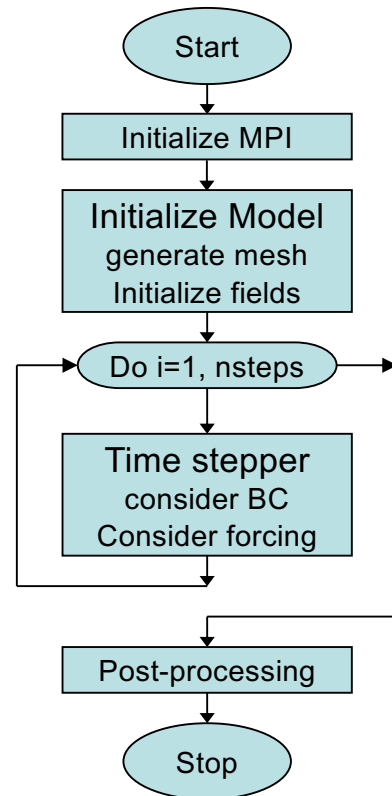
Legend
Model
Extension for data assimilation

PDAF Parallel Data Assimilation Framework

PDAF tutorial – Analysis step in online mode with a parallel model

Fully parallel configuration

- Tutorial shows implementation for a fully parallel case

    ➜ Number of processes equals ensemble size times number of processes used for a single model task!

- For a more flexible (and complicated) configuration see PDAF's online guide

PDAFParallel
Data Assimilation
Framework

model_pdaf: General program structure

```
program main_pdaf

    init_parallel_pdaf       - initialize parallelization

    initialize               - initialize model information

    init_pdaf                - initialize parameters for PDAF
                                  and read ensemble

    integrate                - time stepping loop

        assimilate_pdaf      - compute analysis step
                               (called inside stepping loop)

end program
```

Note:
In the example code, we use different files main.F90 and
main_pdaf.F90 to allow for easy comparison

PDAF Parallel
Data Assimilation
Framework

mod_assimilation.F90

Fortran module

- Declares the parameters used to configure PDAF

- Will be included (with 'use') in the user-written routines

- Additions to template necessary for observation handling

PDAF Parallel
Data Assimilation
Framework

0d) Inserting subroutine calls

PDAF Parallel
Data Assimilation
Framework

Where to insert subroutine calls?

`init_parallel_pdaf`

➜ at the start of the program, but after the MPI_Init performed in the code of the parallel model

`init_pdaf`

➜ after the initialization of the model i.e. directly before the time stepping loop

`assimilate_pdaf`

➜ Last operation in the time stepping loop i.e. just before the 'END DO'

Note: One can add the routines one after the other:
First insert `init_parallel_pdaf` and test the program, then add `init_pdaf`, etc.

PDAF Parallel
Data Assimilation
Framework

init_parallel_pdaf.F90

- It is fully implemented template usable with small adaptions

- Required adaptions

  - Include MPI variables from module of the model:
    MPI_COMM_WORLD, COMM_model, mype_model, npes_model
    (the latter three variables might be named differently in a model)

  - init_parallel_pdaf defines a model communicator
    `comm_model`
    (actually it's a set for communicators, one for each model task)

  - Set communicator of the parallel model to comm_model at the end
    if init_parallel_pdaf:
    "`my_models_communicator`" = `comm_model`
    (include my_models_communicator from module of model)

  - Set variables for number of processes in model and rank of a
    process (`npes_model, mype_model`) at end of routine

PDAF Parallel
Data Assimilation
Framework

init_parallel_pdaf.F90 (2)

- Parallelization variables for PDAF are declared in Fortran module

  `mod_parallel_pdaf`

- Important variable:

  `n_modeltasks`

  - Defines number of concurrent model integrations.

  - Has to be equal to ensemble size

  - In the example: Read as 'dim_ens' from command line (using subroutine 'parse')

- **Important:** If the parallel model uses MPI_COMM_WORLD, this has to be replaced! (MPI_COMM_WORLD denotes always all processes in the program)

PDAF Parallel
Data Assimilation
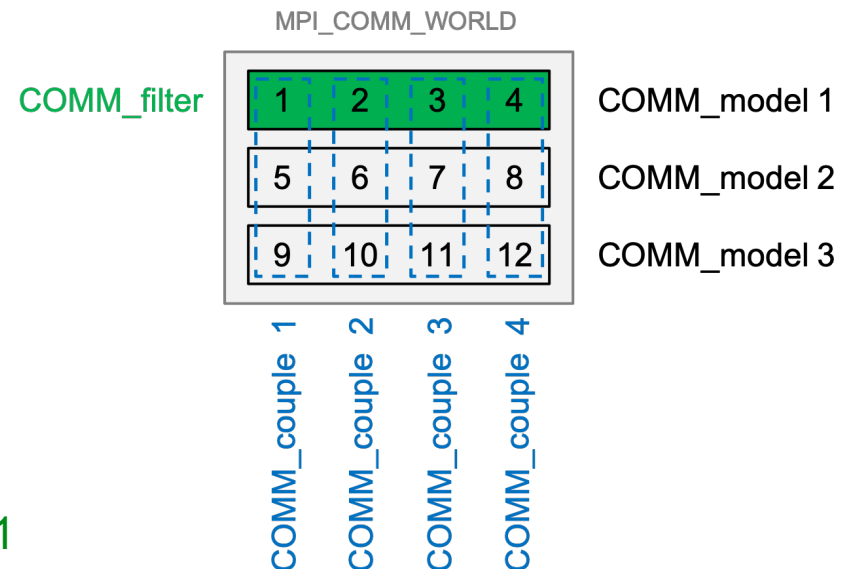Framework

init_parallel_pdaf.F90 (3) - Example

The routine initializes 3 groups of communicators
- COMM_model: Used to run the parallel model forecasts
- COMM_filter: Used to compute the filter
- COMM_couple: Coupling between model and filter processes

These are provided to PDAF when calling PDAF_init

The figure shows an example
- 12 processes in total
- 3 model tasks in parallel
- Each model task uses 4 processes in its COMM_model
- Each COMM_couple links groups of 3 processes to distribute and collect ensemble states
- The filter processes use model task 1

MPI_COMM_WORLD

COMM_filter

| 1 | 2 | 3 | 4 | COMM_model 1
| 5 | 6 | 7 | 8 | COMM_model 2
| 9 | 10 | 11 | 12 | COMM_model 3

COMM_couple 1   COMM_couple 2   COMM_couple 3   COMM_couple 4

init_parallel_pdaf is coded to provide this configuration when running with 12 processes and setting dim_ens=3

PDAF Parallel
Data Assimilation
Framework

init_pdaf.F90

Routine sets parameters for PDAF, calls `PDAF_init`
to initialize the data assimilation, and calls `PDAF_get_state` to
prepare the ensemble integrations:

Template contains list of available parameters
(declared in and used from `mod_assimilation`)

Independent of the filter algorithm:

- Include information on size of model fields from model
- Define dimension of decomposed state vector

$$\text{dim\_state\_p} = \text{nx\_p} * \text{ny}$$

In call to `PDAF_init`, the name of the user-supplied routine for
ensemble initialization routine is specified:
`init_ens_pdaf`

PDAF Parallel
Data Assimilation
Framework

init_pdaf.F90 (II)

In call to PDAF_get_state, the names of 3 user-supplied routines are specified:

```
next_observation_pdaf
```
             - Set number of time steps
              in forecast phase

```
distribute_state_pdaf
```
             - Initialize model fields from state
              vector

```
prepoststep_ens_pdaf
```
             - poststep routine (compute estimated
              errors, write state estimate, etc.)

Initially, one can just copy the template routines. One can adapt them later to the particular application.

PDAF Parallel
Data Assimilation
Framework

assimilate_pdaf.F90

Routine just calls a filter-specific routine like

    `PDAF_assimilate_estkf`

We don't insert `PDAF_assimilate_estkf` directly into the model code

    ➔ because, we need to declare all user-supplied routines as 'EXTERNAL'. This could clutter the model code.

Filter-specific user routines are described next. Initially, one can just copy the template routines.

Note: Template contains calls for `PDAF_assimilate_estkf` and `PDAF_assimilate_lestkf`. Need to adapt for other filters

PDAF Parallel
Data Assimilation
Framework

Differences online and offline

- If you've studied the tutorial for offline mode

<table>
<tr><td align="center">Offline</td><td align="center">Online</td></tr>
<tr><td>

Separate programs for model and assimilation
Needed to implement routine <code>intialize</code>
Grid dimensions declared in <code>mod_assimilation</code>
Ensemble information read from files
<code>mod_assimilation</code> contains all field and assimilation variables

</td><td>

Extend model program for assimilation
Operations in <code>initialize</code> given by model; no changes for assimilation!
Grid dimensions defined in model code (<code>mod_model</code>)
Ensemble information provided by model fields
<code>mod_assimilation</code> only contains variables for assimilation

</td></tr>
</table>

PDAF Parallel
Data Assimilation
Framework

Optional routine: finalize_pdaf.F90

Call to `finalize_pdaf` can be inserted at the end of the model

Routine contains two calls to PDAF_print info:

    `CALL PDAF_print_info(2)`

        – display information on allocated memory inside PDAF

    `CALL PDAF_print_info(1)`

        – display timing information
        (values 3 and 4 also possible for more detailed timers)

Note: finalize_pdaf only prints the information for mype_world==0

In addition there is

    `CALL PDAF_deallocate()`

which deallocates internal arrays in PDAF

PDAF Parallel
Data Assimilation
Framework

0e) Forecast phase

PDAF Parallel Data Assimilation Framework

Files for PDAF

Template contains all required files

> just need to be filled with functionality

init_pdaf.F90

init_ens_pdaf.F90   initialization

next_observation_pdaf.F90

distribute_state_pdaf.F90   ensemble forecast

collect_state_pdaf.F90

init_dim_obs_pdaf.F90

obs_op_pdaf.F90

init_obs_pdaf.F90   analysis step

prodrinva_pdaf.F90

prepoststep_ens_pdaf.F90

PDAF Parallel
Data Assimilation
Framework

init_pdaf.F90

Routine sets parameters for PDAF and calls `PDAF_init` to initialize the data assimilation:

Template contains list of available parameters (declared in and used from `mod_assimilation`)

For the example set :

1. `dim_ens = 9`

2. `rms_obs = sqrt(0.5)`

3. `filtertype = 6` (for ESTKF)

4. `delt_obs = 2` (assimilate afer each 2nd time step)

In call to PDAF_init, the name of the ensemble initialization routine is specified:
   `init_ens_pdaf`

PDAF Parallel
Data Assimilation
Framework

init_ens_pdaf.F90

A *call-back* routine called by PDAF_init:

- Implemented by the user

- Its name is specified in the call to PDAF_init

- It is called by PDAF through a defined interface:

```
SUBROUTINE init_ens_pdaf(filtertype, dim_p,
        dim_ens, state_p, Uinv, ens_p, flag)
```

Declarations in header of the routine shows "intent" (input, output):

```
REAL, INTENT(out)    :: ens_p(dim_p, dim_ens)
```

Note:
All call-back routines have a defined interface and show the intent of the variables. Their header comment explains what is to be done in the routine.

PDAF Parallel
Data Assimilation
Framework

init_ens_pdaf.F90 (2)

Initialize ensemble matrix `ens_p` for the start time of the assimilation

1. Include `nx, ny, nx_p` with `use mod_model`

2. Declare and allocate `real :: field(ny, nx)`

3. Loop over ensemble files `(i=1,dim_ens)`

    for each file:

    - read ensemble state into `field`

    - store local part of `field` in column `i` of `ens_p`
      `(columns nx_p*mype_model+1 : nx_p*mype_model+nx_p)`

4. Deallocate `field`

Note:
Columns of `ens_p` are state vectors. Store following
storage of field in memory (column-wise in Fortran)

PDAF Parallel
Data Assimilation
Framework

The forecast phase

At this point the initialization of PDAF is complete:

- Initial Ensemble of model states is initialized

- Filter algorithm and its parameters are chosen

Next:

- Implement user-routines for forecast phase

- All are call-back routines:

  ➢ User-written, but called by PDAF

Note:
Some variables end with _p.
It means that the variable is specific for a process
(its values are different for each process)

PDAF Parallel
Data Assimilation
Framework

next_observation_pdaf.F90

Routine to

- Set number of time steps in next forecast phase

- Set flag to control exit from forecasts (`doexit`)

Most simple setting:

include `delt_obs` from `mod_assimilation`

`nsteps = delt_obs`

`doexit = 0`

Note: The assimilation program stops when the maximum number of time steps of the model is reached, even if doexit=0

PDAF Parallel
Data Assimilation
Framework

More sophisticated setting:

- Utilize `stepnow` (current time step) and `total_steps` (total number of time steps given by model).

```
IF (stepnow + nsteps <= total_steps) THEN
      nsteps = delt_obs        ! Forecast length
      doexit = 0               ! Continue assimilation
ELSE
      nsteps = 0               ! No more steps
      doexit = 1               ! Exit assimilation
END IF
```

Note: In the example `doexit=1` is used only inside PDAF and avoids some screen output.

PDAF Parallel
Data Assimilation
Framework

distribute_state_pdaf.F90

Routine to

- Initialize model fields from a state vector

- Routine is provided with the state vector `vector_p`

For the example:

1. Access `nx_p, ny` and `field_p` with `use mod_model`

2. Initialize model field from state vector:

```
DO j = 1, nx_p
   field_p(1:ny, j) = state_p(1+(j-1)*ny : j*ny)
END DO
```

PDAF Parallel
Data Assimilation
Framework

prepoststep_ens_pdaf.F90

Post-step routine for the online mode:

Already there in the template:

1. Compute ensemble mean state `state_p`
2. Compute estimated variance vector `variance`
3. Compute estimated root mean square error `rmserror_est`

Possible extensions:

4. Write analysis state (ensemble mean, `state_step*_ana.txt`)

5. Write analysis ensemble into files
   (Analogous to reading in `init_ens_pdaf`)

6. Analogously one can write the forecast fields

PDAFParallel
Data Assimilation
Framework

Completion of forecast phase

At this point the implementation of the forecast phase is practically complete:

- Initial ensemble and PDAF's parameters are set

- The ensemble forecast can be computed

One can now compile the program model_pdaf (make model_pdaf) to check whether it runs.

Note: It is recommended to compile PDAF with –DPDAF_NO_UPDATE at this point as the routine for the analysis step are not yet implemented.

Note: For now, `prepoststep_ens_pdaf` only lets you test the initial ensemble. Testing the forecast fields need implementation of routine `collect_state_pdaf`

PDAF Parallel
Data Assimilation
Framework

1a) Global filter

PDAF Parallel
Data Assimilation
Framework

Running the tutorial program

- **cd** `to /tutorial/classical/online_2D_serialmodel`

- Set environment variable `PDAF_ARCH` or set it in Makefile
  (e.g. `linux_gfortran_openmpi`)

- Compile by running `'make model_pdaf'`
  (next slide will discuss possible compile issues)

- Run the program with
  `mpirun -np 18 ./model_pdaf -dim_ens 9`

- Inputs are read in from `/tutorial/inputs_online`

- Outputs are written in
  `/tutorial/classical/online_2D_parallelmodel`

- Plot result, e.g. with 'octave':

  `load state_step10_ana.txt`

  `pcolor(state_step10_ana)`

PDAF Parallel
Data Assimilation
Framework

Requirements for compiling PDAF

PDAF requires libraries for BLAS and LAPACK

- Libraries to be linked are specified in the include file for make
  in `/make.arch` (file according to `PDAF_ARCH`)

- For `$PDAF_ARCH=linux_gfortran_openmpi` the specification is

  `LINK_LIBS =-L/usr/lib -llapack  -lblas   -lm`

- If the libraries are at another non-default location, one has to change
  the directory name (`/usr/lib`)

- Some systems or compilers have special libraries
  (e.g. MKL for ifort compiler, or ESSL on IBM/AIX)

PDAF needs to be compiled for double precision

- Needs to be set at compiler time in the include file for make:

- For gfortran: `OPT = -O3 -fdefault-real-8`

PDAF Parallel
Data Assimilation
Framework

Files in the tutorial implementation

`/tutorial/inputs_online`

- `true_stepY.txt`   true state

- `state_ini.txt`   initial estimate (ensemble mean)

- `obs_stepY.txt`   observations

- `ens_X.txt`   initial ensemble members

`/tutorial/classical/online_2D_parallelmodel`

(after running model_pdaf)

- `state_stepY_ana.txt`   analysis state estimate

- `ens_X_stepY_ana.txt`   analysis ensemble members

X=1,…,9: ensemble member index

Y=1,…,18: time step index

Note: Files *_for.txt contain forecast fields

PDAF Parallel
Data Assimilation
Framework

Result of the global assimilation

For example, at step 10

- The analysis state (center) is closer to the true field than without assimilation (left)

- Truth and analysis are nearly identical (right)

PDAF Parallel
Data Assimilation
Framework
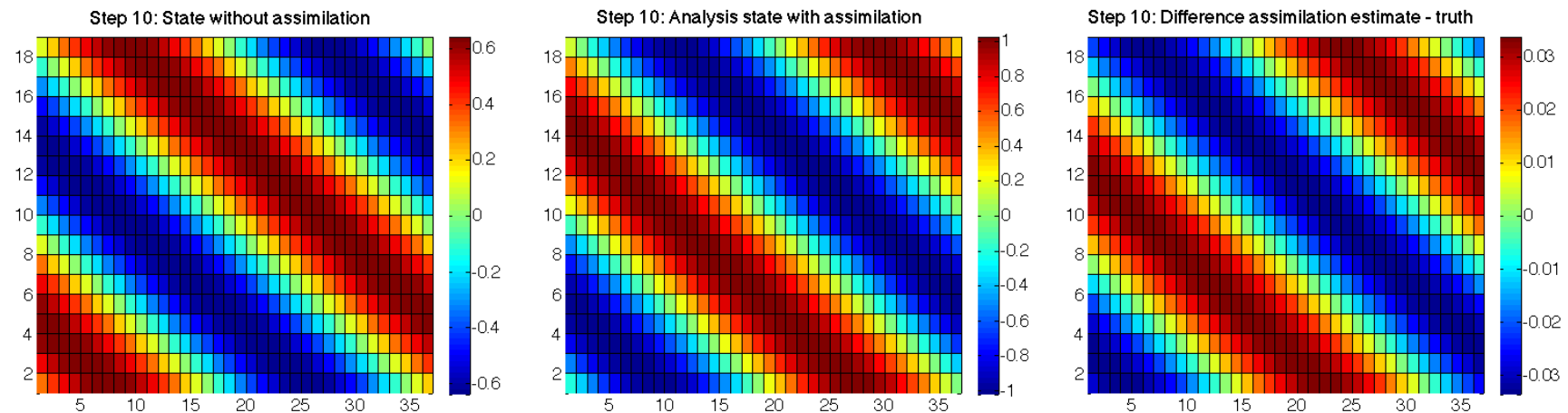
The analysis step

Next: Implement user-routines for the analysis step

The analysis step needs several user-supplied routines
for operations like

- write forecast model fields into state vector

- determine number of available observations

- observation operator acting on a state vector

- initialization of the vector of observations

PDAF Parallel
Data Assimilation
Framework

collect_state_pdaf.F90

Routine to

- Fill state vector with forecasted model fields

- Routine is provided with the state vector `vector_p`

For the example:

1. Access `nx, ny` and `field` with `use mod_model`

2. Initialize state vector from model field:

```
DO j = 1, nx_p
  state_p(1+(j-1)*ny : j*ny) = field_p(1:ny, j)
END DO
```

Note: The routine is independent of the filter!

PDAF Parallel
Data Assimilation
Framework

init_dim_obs_pdaf.F90

Routine to

- read observation file

- Count number of observations for process-local part
  of state vector (direct output to PDAF: `dim_obs_p`)

Optional, also

- Initialize array holding process-local available
  observations

- Initialize index array telling index of observation point
  in process-local state vector

The most complicated routine in the example!
(but only 123 lines)

PDAF Parallel
Data Assimilation
Framework

Preparations and reading of observation file:

1. Include `nx, ny, nx_p` with `use mod_model`

2. declare and allocate real array `obs_field(ny, nx)`

3. Get offset of local part in global state vector

   `off_p` = Sum over nx_p*ny up to i=mype_filter

4. read observation file for current time step:

Initialize string 'stepstr' for time step

```
OPEN (12, &
    file='inputs_online/obs'//stepstr//'.txt', &
    status='old')
DO i = 1, ny
    READ (12, *) obs_field(i, :)
END DO
CLOSE (12)
```

PDAF Parallel
Data Assimilation
Framework

Count available process-local observations (**dim_obs_p**):

1. Declare `integer :: cnt0, cnt_p`

2. Now count

```
cnt0 = 0
cnt_p = 0
DO j = 1, nx
  DO i= 1, ny
    cnt0 = cnt0 + 1
    IF (cnt0>off_p .AND. cnt0<=off_p+nx_p*ny) THEN
       IF (obs_field(i,j) > -999.0) cnt_p = cnt_p + 1
END IF; END DO; END DO
dim_obs_p = cnt_p
```

PDAF Parallel
Data Assimilation
Framework

init_dim_obs_pdaf.F90 (4)

Initialize observation vector (`obs_p`)
    and index array (`obs_index_p`):

1. Include `obs_p` and `obs_index_p`
    with `use mod_assimilation`

2. Allocate
    `obs_p(dim_obs_p), obs_index_p(dim_obs_p)`
    (If already allocated, deallocate first)

3. Now initialize …

Note:
The arrays only contain information about valid observations;
one could store observations already in files in this way.

PDAF Parallel
Data Assimilation
Framework

Initialize `obs` and `obs_index`

```
cnt0 = cnt_p = cnt0_p = 0    ! Count grid points
DO j = 1, nx
  DO i= 1, ny
    cnt0 = cnt0 + 1
    IF (cnt0>off_p .AND. &
        cnt0<=off_p+nx_p*ny) THEN
      cnt0_p = cnt0_p + 1
    IF (obs_field(i,j) > -999.0) THEN
      cnt_p = cnt_p + 1
      obs_index_p(cnt_p) = cnt0_p     ! Index
      obs_p(cnt_p) = obs_field(i, j)  ! observations
    END IF; END IF
  END DO
END DO
```

PDAF Parallel
Data Assimilation
Framework

obs_op_pdaf.F90

Implementation of observation operator
  acting one some state vector

Input: state vector `state_p`

Output: observed state vector `m_state_p`

1. Include `obs_index_p` by `use mod_assimilation`

2. Select observed grid points from state vector:

```
DO i = 1, dim_obs_p
  m_state_p(i) = state_p(obs_index_p(i))
END DO
```

Note:
`dim_obs_p` is an input argument of the routine

PDAF Parallel
Data Assimilation
Framework

init_obs_pdaf.F90

Fill PDAF's observation vector

Output: vector of observations `observation_p`

1. Include `obs_p` with `use mod_assimilation`

2. Initialize `observation_p`:

   `observation_p` `= obs_p`

Note:
This is trivial, because of the preparations in `init_dim_obs_pdaf`!

(However, the operations needed to be separate, because PDAF allocates `observations_p` after the call to `init_dim_obs_pdaf`)

PDAF Parallel
Data Assimilation
Framework

prodrinva_pdaf.F90

Compute the product of the inverse observation error covariance matrix with some other matrix

- Input: Matrix `A_p(dim_obs_p, rank)`
- Output: Product matrix **`C_p`**`(dim_obs_p, rank)`
  (`rank` is typically `dim_ens-1`)

1. Declare and initialize inverse observation error variance

   ```
   ivariance_obs = 1.0 / rms_obs**2
   ```

2. Compute product:

   ```
   DO j = 1, rank
     DO i = 1, dim_obs_p
        C_p(i, j) = ivariance_obs * A_p(i, j)
     END DO
   END DO
   ```

PDAF Parallel
Data Assimilation
Framework

Done!

The analysis step in online mode with the parallelized model is fully implemented now

The implementation allows you now to use the global filter ESTKF (ETKF and SEIK are usable by adding a call to the corresponding routines PDAF_assimilate_X in assimilate_pdaf)

Not usable are EnKF and SEEK (The EnKF needs some other user files und SEEK a different ensemble initialization)

PDAF Parallel
Data Assimilation
Framework

A complete analysis step

We now have a fully functional analysis step
- if no localization is required!

Possible extensions for a real application:

Adapt routines for

➢ Multiple model fields

    ➔ Store full fields consecutively in state vector

➢ Third dimension

    ➔ Extend state vector

➢ Different observation types

    ➔ Store different types consecutively in observation vector

➢ Other file type (e.g. binary or NetCDF)

    ➔ Adapt reading/writing routines

PDAF Parallel
Data Assimilation
Framework

Differences between online and offline modes

For the analysis step in online mode:

`collect_state_pdaf`   - additional routine for online mode

`init_dim_obs_pdaf`   - read from file for current time step; include `nx, ny` from `mod_model` instead of `mod_assimilate`

`obs_op_pdaf`   - identical in online and offline modes

`init_obs_pdaf`   - identical in online and offline modes

`prodrinva_pdaf`   - identical in online and offline modes

PDAF Parallel
Data Assimilation
Framework

1b) Local filter with parallelized model

PDAF Parallel
Data Assimilation
Framework

Localization

Localization is usually required for high-dimensional systems

- Update small regions (S)
  (e.g. single grid points, single vertical columns)

- Consider only observations within cut-off distance (D)

- Weight observations according to distance from S

PDAF Parallel
Data Assimilation
Framework
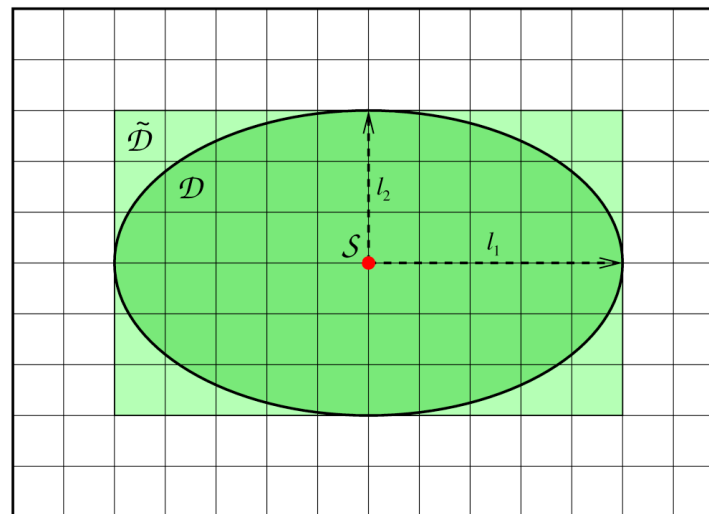
The FULL observation vector

- A single local analysis at *S* (single grid point) need observations from domain *D*

- A loop of local analyses over all *S* needs all observations

  - This defines the *full* observation vector

- Why distinguish *full* and *all* observations?

  ➜ They can be different in case of parallelization!

  - Example:

    ➢ Split domain in left and right halves

    ➢ Some of the analyses in left half
       need observations from the right side.

    ➢ Depending on localization radius not all observations from
       the right side might be needed for the left side analyses

PDAF Parallel
Data Assimilation
Framework

Running the tutorial program

- Compile as for the global filter
- Run the program with

  `mpirun -np 18 ./model_pdaf -dim_ens 9 OPTIONS`

- `OPTIONS` are always of type `-KEYWORD VALUE`
- Possible `OPTIONS` are

  `-filtertype 7`   (select LESTKF if not set in init_pdaf)

  `-cradius 5.0`   (set localization radius, 0.0 by default, any
                      positive value should work)

  `-locweight 2`   (set weight function for localization, default=0
                      for constant weight of 1; possible are integer
                      values 0 to 4; see `init_pdaf`)

  Note: You can run the model e.g. using 18 MPI-processes even on most
  computers with only 2 processor cores. However, to see a speedup in
  computing time, you need more physical processors

PDAF Parallel
Data Assimilation
Framework

Result of the local assimilation

`mpirun -np 9./model_pdaf -dim_ens 9 -filtertype 7`

- Default: zero localization radius (cradius=0.0)

- Change only at observation locations

PDAF Parallel
Data Assimilation
Framework
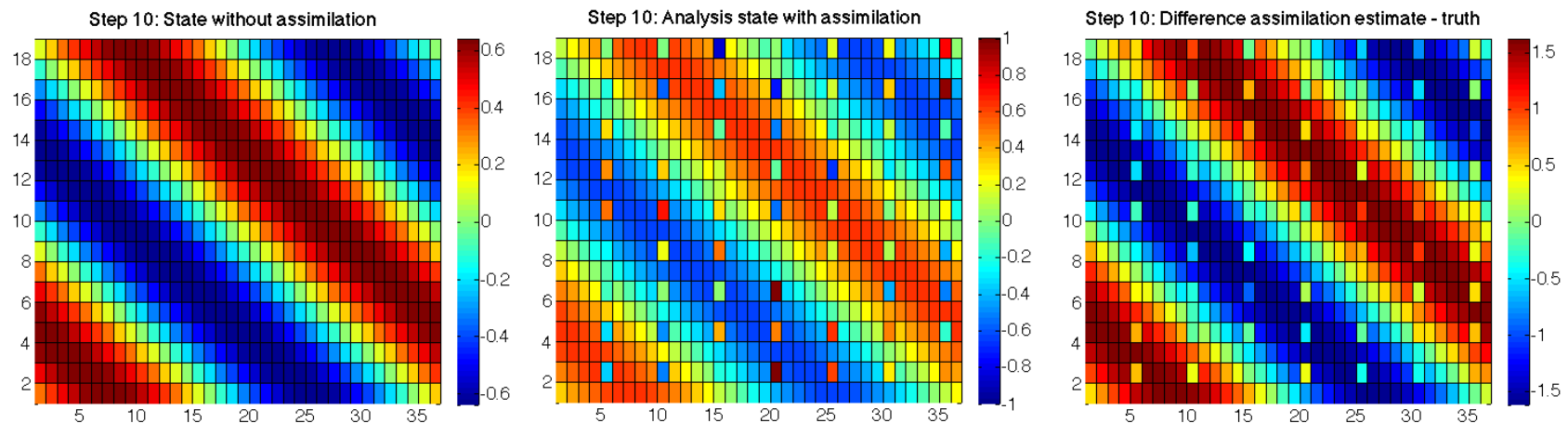
Result of the local assimilation (2)

… `-filtertype 7` `-cradius 10.0`

- All local analysis domains are influenced (all see observations)

- Up to 16 observations in a single local analysis (average 9.6)

Note: The set up of the experiment favors the global filter because of the shape of the ensemble members

PDAF Parallel
Data Assimilation
Framework
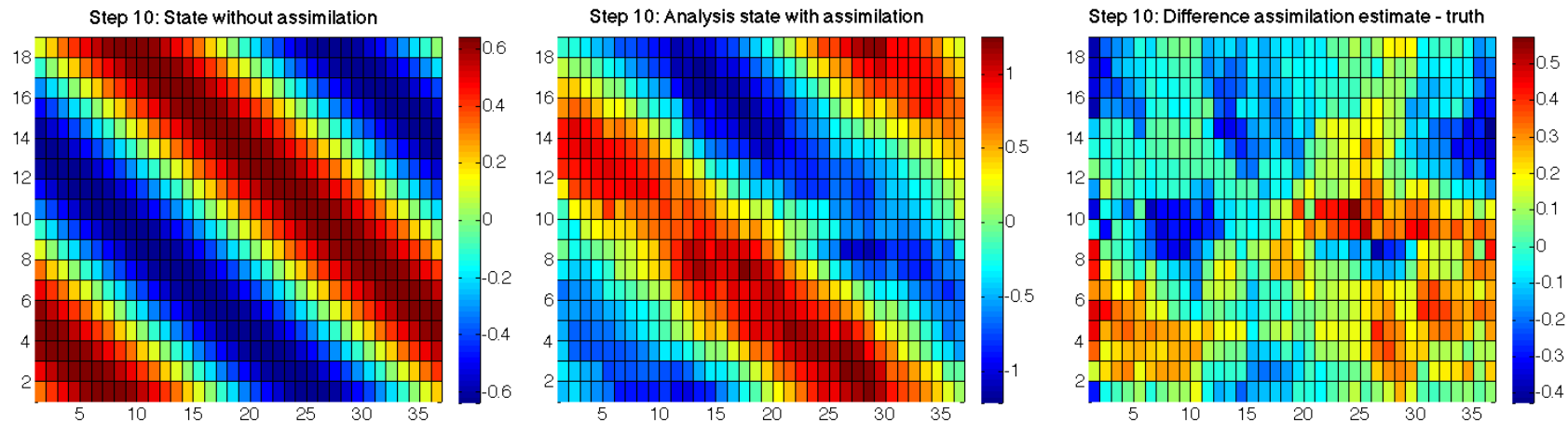
Result of the local assimilation (2)

`… -filtertype 7 -cradius 10.0 -locweight 2`

- Observation weighting by 5th-order polynomial

- Analysis field is smoother than before (because of weighting)

PDAF Parallel
Data Assimilation
Framework
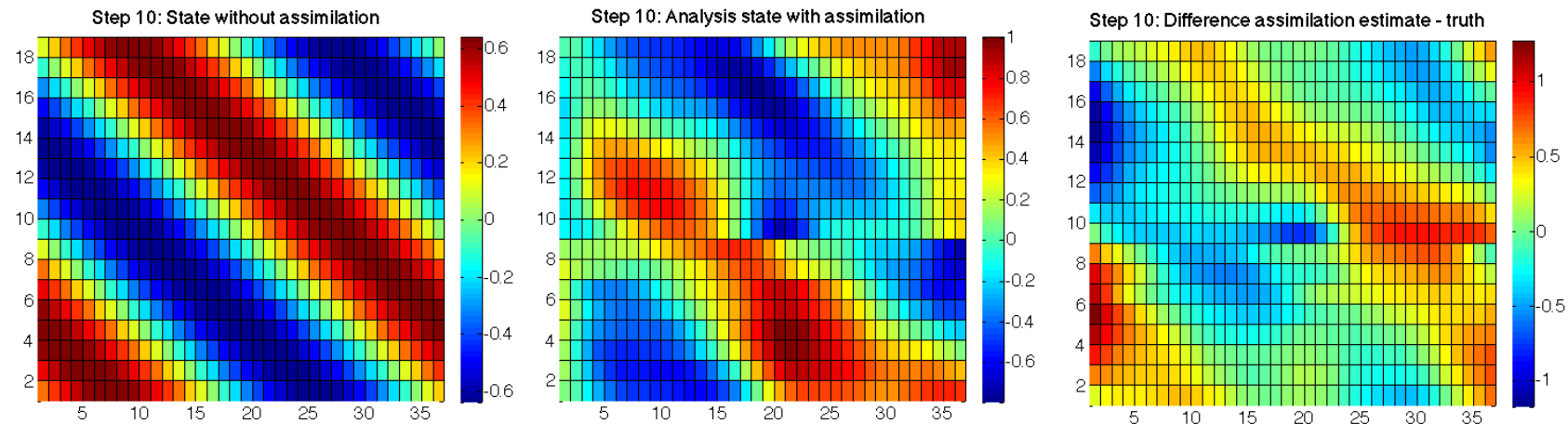
Result of the local assimilation (3)

```
… -filtertype 7 -cradius 40.0
```

- Large radius: All local analysis domains see all observations

- Result identical to global filter

PDAF Parallel
Data Assimilation
Framework
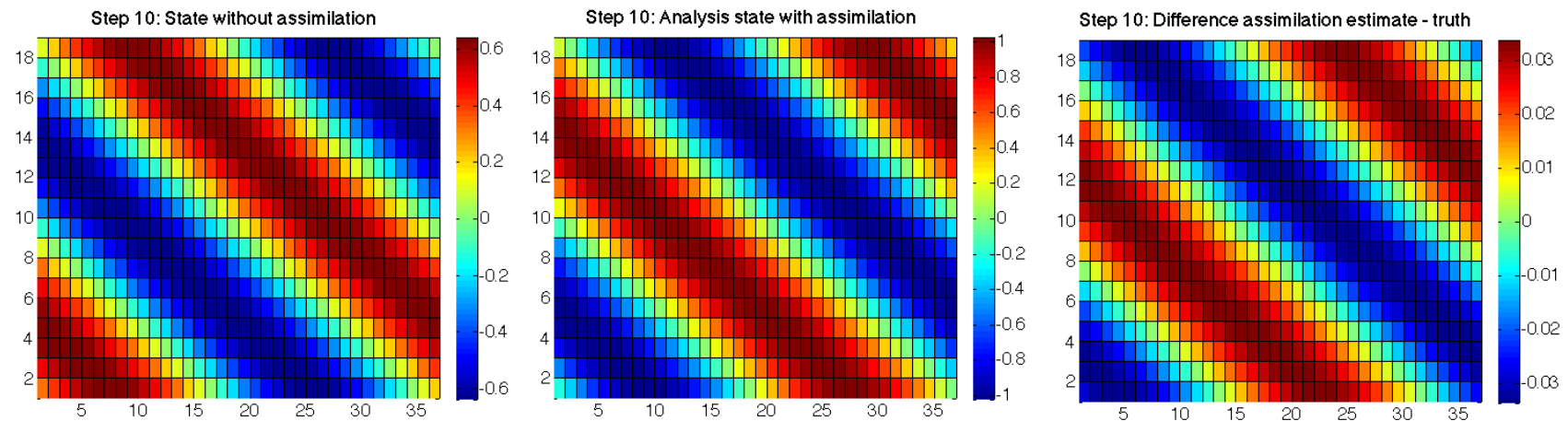
Local filter LESTKF

- Localized filters are a variant of the global filters

- User written files for global filter can be widely re-used

- Additional user-written files to handle local part

- No changes to:

    initialize.F90

    init_ens_pdaf.F90

    prepoststep_ens_pdaf.F90

- Change in init_pdaf.F90:

    Set filtertype = 7

    (You can also set it later on command line)

PDAF Parallel
Data Assimilation
Framework

Local filter LESTKF (2)

Adapt files from global analysis

init_dim_obs_pdaf.F90 ➜ init_dim_obs_**f**_pdaf.F90

obs_op_pdaf.F90 ➜ obs_op_**f**_pdaf.F90

init_obs_pdaf.F90 ➜ init_obs_**f**_pdaf.F90

prodrinva_pdaf.F90 ➜ prodrinva_**l**_pdaf

Naming scheme:

_f_   "full": operate on all required observations
          (without parallelization these are all observations)

_l_   "local": operation in local analysis domain or corresponding
          local observation domain

PDAF Parallel
Data Assimilation
Framework

Local filter LESTKF (3)

Additional files for local analysis step

```
init_n_domains_pdaf.F90
init_dim_l_pdaf.F90         localize
g2l_state_pdaf.F90          state vector
l2g_state_pdaf.F90
init_dim_obs_l_pdaf.F90     localize
g2l_obs_pdaf.F90            observations
init_obs_l_pdaf.F90
```

Discuss now the files in the order they are called

PDAF Parallel
Data Assimilation
Framework

init_n_domains_pdaf.F90

Routine to set the number of local analysis domains

Output: `n_domains_p`

For the example: number of process-local grid points (nx_p * ny)

To do:

1. Include `nx_p, ny` with `use mod_model`
2. Set

$$\textbf{n\_domains\_p} = nx\_p * ny$$

PDAF Parallel
Data Assimilation
Framework

init_dim_obs_f_pdaf.F90

Operations in case of parallelization:

- Read observation file

- Count number of observations for process-local part of state vector (`dim_obs_p`)

- Initialize arrays holding process-local available observations (`obs_p`) and their coordinates (`coords_obs_p`)

- Initialize index array (`obs_index_p`) telling index of a process-local observation in process-local state vector

- Initialize full number of observations (`dim_obs_f`), vector of observations (`obs_f`), and coordinates (`coords_obs_f`)

"FULL" observation vector:
All observations required for all local analyses in process-local part of state vector (Here: *Full=All* observations for simpicity)

Adapt `init_dim_obs_pdaf` from global filter …

PDAF Parallel
Data Assimilation
Framework

init_dim_obs_f_pdaf.F90 (2)

Initialize coordinates of process-local observations

For the local filter:

1. Copy functionality from `init_dim_obs_pdaf.F90`
   (In the subroutine definition take care that dim_obs_f is used instead of dim_obs_p)

2. Include `dim_obs_p` with `use mod_assimilation`

3. Initialization of observation coordinates

   a) Add allocatable REAL arrays `coords_obs_p` and `obs_p` to the routine (don't include `obs_p` from module)

   b) In the loop where `obs_p` is set add
   ```
   coords_obs_p(1, cnt_p) = REAL (j)
   coords_obs_p(2, cnt_p) = REAL (i)
   ```

PDAF Parallel
Data Assimilation
Framework

init_dim_obs_f_pdaf.F90 (3)

Initialize full quantities (`dim_obs_f, obs_f, coords_obs_f`)

1. Include allocatable arrays `coords_obs_f` and obs_f
   with `mod_assimilation`

2. Obtain `dim_obs_f` by calling `PDAF_gather_dim_obs_f`

3. Allocate `obs_f` and `coords_obs_f`
   (deallocate first if already allocated)

4. Obtain `obs_f` by calling `PDAF_gather_obs_f`

5. Obtain `coords_obs_f` by calling `PDAF_gather_obs_f`

6. Add DEALLOCATE for `obs_p` and `coords_obs_p`

Note: It is mandatory to call PDAF_gather_dim_obs_f once before using the two other functions because it stores dimension information.

Note: The three PDAF functions have been added with PDAF Version 1.13 to avoid that the user implementation needs calls to MPI functions.

Note: `coords_obs_f` has to be a REAL array

PDAF Parallel
Data Assimilation
Framework

obs_op_f_pdaf.F90

Implementation of observation operator
    for full observation domain

Difficulty:

- The state vector `state_p` is local to each process

- Full observed vector goes beyond process boundary

Implement two steps:

1. Initialize process-local observed state

2. Gather full observed state vector using MPI

PDAF Parallel
Data Assimilation
Framework

obs_op_f_pdaf.F90 (2)

1. Initialize process-local observed state `m_state_p`

   a) Include `dim_obs_p` and `obs_index_p`
      with `use mod_assimilation`

   b) Declare real allocatable array `m_state_p(:)`

   c) Allocate
      `m_state_p(dim_obs_p)`

   d) Fill the array

   ```
   DO i = 1, dim_obs_p
     m_state_p(i) = state_p(obs_index_p(i))
   END DO
   ```

PDAF Parallel
Data Assimilation
Framework

obs_op_f_pdaf.F90 (3)

2. Get full observed state vector

a) Add variable `INTEGER :: status`

b) Add call to PDAF_gather_obs_f:

`CALL PDAF_gather_obs_f(m_state_p, m_state_f, status)`

c) Deallocate m_state_p

Note: It is mandatory to call `PDAF_gather_dim_obs_f` once before using the two other functions because it stores dimension information. Usually this was already done in `init_dim_obs_f_pdaf`

PDAF Parallel
Data Assimilation
Framework

init_dim_l_pdaf.F90

Set the vector size `dim_l` of the local analysis domain

Further set the coordinates of the local analysis domain and the indices of the elements of the local state vector in the global state vector

Each single grid point is a local analysis domain in the example

1. Set **dim_l** = 1

PDAF Parallel
Data Assimilation
Framework

2. Determine coordinates of local analysis domain

1. Compute offset:

`off_p` = Sum over `nx_p*ny` up to i=mype_filter

2. Include `coords_l` with `use mod_assimilation`

3. Include `nx, ny, nx_p` with `use mod_model`

4. Compute `coords_l` from `nx, ny`:

```
coords_l(1)
  = real(ceiling(real(domain_p + off_p)/real(ny)))
coords_l(2)
  = real(domain_p + off_p) - (coords_l(1)-1)*ny
```

Note: With parallelization the domain numbering begins with 1 for each process. For the coordinates we also need to count the domains from processes with lower process rank using `off_p`

PDAF Parallel
Data Assimilation
Framework

3. Set indices of the elements of the local state vector in the global decomposed state vector

a) Include `id_lstate_in_pstate`
   with `use mod_assimilation`

b) Allocate `id_lstate_in_pstate(dim_l)`
   (Deallocate first if already alloced)

c) Specify the index: It's identical to `domain_p` here
   (because we only have a single model variable)

   ```
   id_lstate_in_pstate(1) = domain_p
   ```

PDAF Parallel
Data Assimilation
Framework

init_dim_obs_l_pdaf.F90

Set the size of the observation vector for the local analysis domain

As for the global filter, this is the longest routine (~102 lines)

Only direct output: `dim_obs_l`

Operations:

1. Include coordinates `coords_l` with `use mod_assimilation`

2. Determine coordinate range for observations

3. Count observations within prescribed localization radius

4. Set index array for local observations (`id_lobs_in_fobs`) and array of distances of local observations (`distance_l`)

Note: The index array in step 4 is re-used for an efficient implementation of g2l_obs_pdaf. The local distance array initialized in step 4 is re-used in prodrinva_l_pdaf avoiding to recompute distances.

PDAF Parallel Data Assimilation Framework

init_dim_obs_l_pdaf.F90 (2)

2. Determine coordinate range for local observations

   1. Declare `real :: limits_x(2), limits_y(2)`

   2. Include `cradius` with `use mod_assimilation`

   3. Set lower and upper limits. E.g. for x-direction

```
limits_x(1) = coords_l(1) - cradius
if (limits_x(1) < 1.0) limits_x(1) = 1.0
limits_x(2) = coords_l(1) + cradius
if (limits_x(2) > real(nx)) limits_x(2) = real(nx)
```

   (analogous for y-direction)

Note: Using `limits_x, limits_y` is not strictly required, but it makes the search for local observations more efficient.

If the localization is only based on grid point indices, the coordinates could be handled as integer values

PDAF Parallel

Data Assimilation

Framework

3. Count local observations (within distance `cradius`)

```
dim_obs_l = 0
DO i = 1, dim_obs_f
   IF ("coords_obs_f(:,i) within coordinate limits") THEN
      Compute distance between coords_obs and coords_l
      IF (distance <= cradius) &
            dim_obs_l = dim_obs_l + 1
   END IF
END DO
```

Note:
For efficiency, we only compute `distance` for observations within coordinate limits `limits_x, limits_y`. Valid local observations reside within circle of radius `cradius`.

PDAF Parallel
Data Assimilation
Framework

4. Set index array for local observations

   ➢ Index of a local observation in the full observation vector

   1. Include `id_obs_in_fobs` and `distance_l` with `use mod_assimilation`

   2. Allocate `id_obs_in_fobs(dim_obs_l)`

   3. Fill index array:

```
cnt = 0
DO i = 1, dim_obs_f
  IF ("coords_obs(:,i) within coordinate limits") THEN
    Compute distance between coords_obs and coords_l
    IF (distance <= cradius) THEN
      cnt = cnt + 1
      id_lobs_in_fobs(cnt) = i
      distance_l(cnt) = distance
END …
```

PDAF Parallel
Data Assimilation
Framework

g2l_state_pdaf.F90 & l2g_state_pdaf.F90

g2l_state_pdaf: Initialize state vector for local analysis domain
from global state vector

l2g_state_pdaf: Initialize global state vector
from state vector for local analysis domain

➢ The templates provide a generic implementation
using the array `id_lstate_in_fstate`

➜ We use the templates without any changes!

> **Note:** The **PDAFlocal** module introduced in PDAF 2.3 allows to implement without using these two routines. Please see the tutorial slides for the offline implementation with PDAF-OMI for a description on how to use the PDAFlocal routines. It can be used independently from PDAF-OMI.

PDAF Parallel
Data Assimilation
Framework

g2l_obs_pdaf.F90 & init_obs_l_pdaf.F90

g2l_obs_pdaf: Initialize local observed state vector from full observed vector

init_obs_l_pdaf: Initialize local vector of observations

> ➤ The templates provide a generic implementation using the array `id_lobs_in_fobs`

➜ We use the templates with out any changes!

Note:

`init_obs_l_pdaf` requires that the full observation vector is stored in the array `obs_f`

PDAF Parallel
Data Assimilation
Framework

prodrinva_l_pdaf.F90

Compute the product of the inverse observation error covariance matrix with some other matrix

+ apply observation localization (weighting)

➢ The weighting and the product are fully implemented for a diagonal observation error covariance matrix with constant variance

➜ When we re-use the array `distance_l` initialized in `init_dim_obs_l_pdaf`, the template can be used without changes.

PDAF Parallel
Data Assimilation
Framework

Done!

Now, the analysis step for local ESKTF in offline mode is fully implemented.

The implementation allows you now to use the local filter LESTKF (LETKF, LSEIK can be used after adding calls to PDAF_assimilate_X)

Not usable are EnKF and SEEK (PDAF does not have localization for these filters)

For testing one can vary localization parameters:

`cradius` – the localization cut-off radius

`locweight` – the weighting method

Default are `cradius=0.0` (observation at single grid point) and `locweight=1` (uniform weight)

PDAF Parallel
Data Assimilation
Framework

2) Hints for adaptions for real models

PDAF Parallel
Data Assimilation
Framework

Implementations for real models

- Tutorial demonstrates implementation for simple model

- You can base your own implementation on the tutorial implementation or the templates provided with PDAF

- Need to adapt most routines, e.g.

  - Specify model-specific state vector and its dimension

  - Adapt `distribute_state` and `collect_state`

  - Adapt routines handling observations

- Further required changes

  - Adapt file output (usually only want to write ensemble mean state in `prepoststep_pdaf`; sometimes possible to use output routines from model)

PDAFParallel
Data Assimilation
Framework

Multiple fields in state vector

- Tutorial uses a single 2-dimensional field

- All fields that should be updated by the assimilation have to be part of the state vector

- For more fields:

  - concatenate them in the state vector

  - adapt state dimension in `init_pdaf`

  - adapt `init_ens_pdaf, collect_state_pdaf, distribute_state_pdaf, prepoststep_pdaf`

  - For local filters: Adapt full (`_f_`) and local (`_l_`) routines and `g2l_state_pdaf, l2g_state_pdaf, g2l_obs_pdaf`

- **Note**

  - It can be useful to define a vector storing the offset (position) of each field in the state vector

  > Note: The tutorial for PDAF-OMI includes an example code using 2 model fields

PDAF Parallel
Data Assimilation
Framework

Multiple observed fields

- In tutorial: observed one field at some grid points

- For several observed fields adapt observation routines:

    - concatenate observed fields in observation vector

    - adapt all observation-handling routines

- **Note**

    - The observation errors can be set differently for each observed field (e.g. using an array `rms_obs`)

    - The localization radius can be set specific for each observed field (observation search in `init_dim_obs_l_pdaf` would use different `cradius` for different fields)

    - One can use spatially varying observation errors using an array `rms_obs` in `prodrinva(_l)_pdaf`

> Note: Using the PDAF-OMI functionality makes the handling of multiple observation types much easier. See the tutorial slides for PDAF-OMI on how to use this.

PDAF Parallel
Data Assimilation
Framework

The End!

Tutorial described example implementations

- Online mode of PDAF

- Simple 2D model with parallelization

- Parallelization over ensemble members at the model itself

- Square root filter ESTKF

    - global and with localization

- Extension to more realistic cases possible with limited coding

- Applicable also for large-scale problems

For full documentation of PDAF
and the user-implemented routines
see http://pdaf.awi.de

PDAF Parallel
Data Assimilation
Framework