PDAF Tutorial

Implementation of the analysis step

in online mode with a serial model

using PDAF-OMI

http://pdaf.awi.de

PDAF Parallel
Data Assimilation
Framework

Implementation Tutorial for PDAF online with serial model

We demonstrate the implementation

of an online analysis step with PDAF

with a model that is itself not parallelized

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 V2.1 and later)

PDAF Parallel
Data Assimilation
Framework

Implementation Tutorial for PDAF online / serial model

This is just an example!

For the complete documentation of PDAF's interface

see the documentation

at http://pdaf.awi.de

PDAFParallel
Data Assimilation
Framework

Overview

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

2 Parts

a) Global filter   b) Localized filter

(and OpenMP-parallelization)

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 serial model. The implementation with a parallelized model is described in a separate tutorial.

PDAFParallel
Data Assimilation
Framework

Contents

PDAF Parallel
Data Assimilation
Framework

PDAF tutorial – Analysis step in offline mode with a serial model

0a) Files for the Tutorial

Tutorial implementation

Files are in the PDAF package

Directory:

`/tutorial/online_2D_serialmodel`

- 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'.

PDAFParallel
Data Assimilation
Framework

Template files for online mode

Directory: `/templates/online_omi`

- 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

PDAFParallel
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)
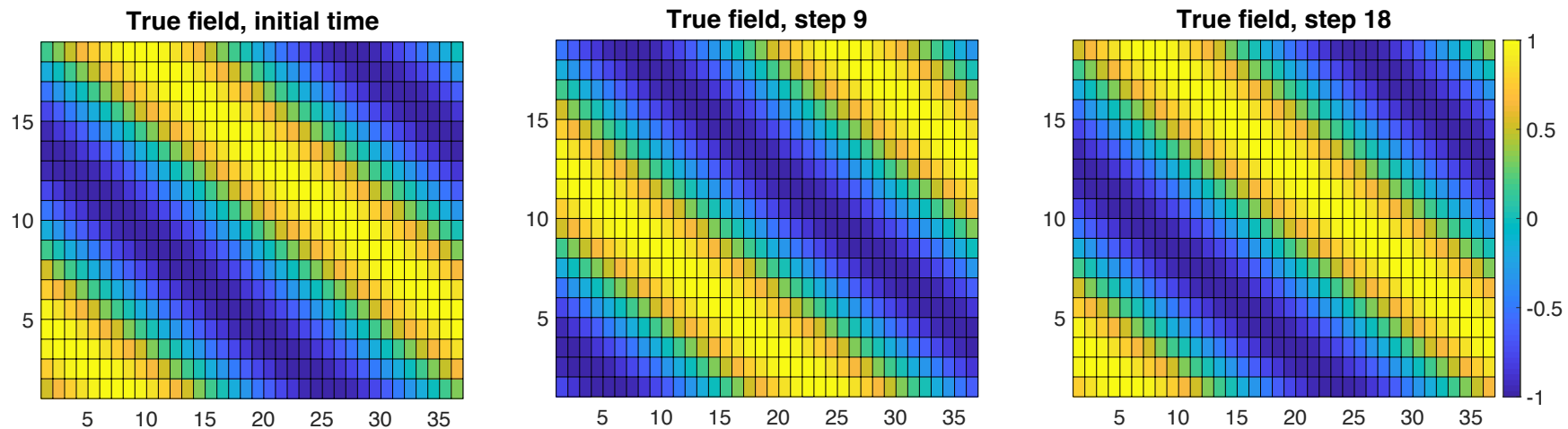
`$PDAF_ARCH`

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

PDAF Parallel
Data Assimilation
Framework

0b) The model without parallelization

PDAF Parallel
Data Assimilation
Framework

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

Model: Files

The model source code consists of the following files:

- mod_model.F90

- main.F90

- initialize.F90

- integrate.F90

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

PDAF Parallel
Data Assimilation
Framework

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

PDAFParallel
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

PDAFParallel
Data Assimilation
Framework

0d) PDAF online mode

PDAF Parallel
Data Assimilation
Framework

Online mode

- Combine model with PDAF into single program

  - "`model_pdaf`"

- Add 3 subroutine calls:

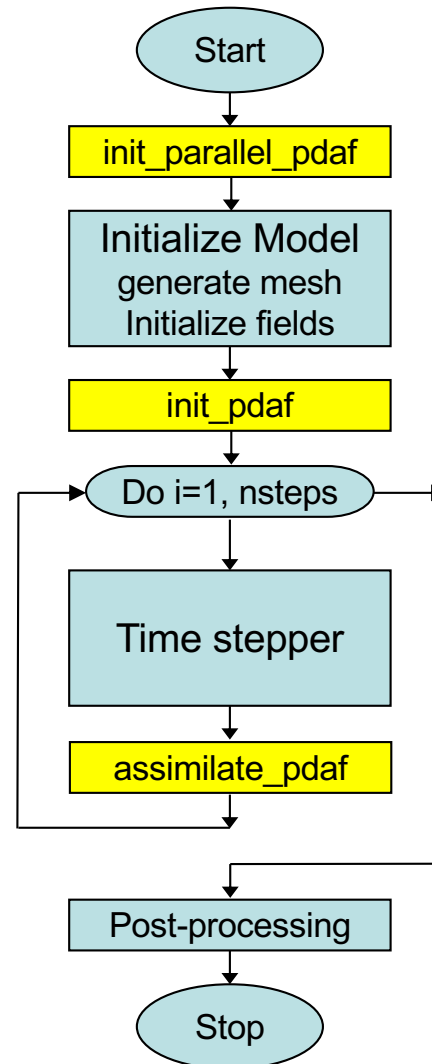 | | |
  |---|---|
  | `init_parallel_pdaf` | - add parallelization |
  | `init_pdaf` | - initialize assimilation |
  | `assimilate_pdaf` | - perform assimilation |

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

  - observation operator

  - initialization of observation vector

  - transfer between state vector and model fields

PDAFParallel
Data Assimilation
Framework

Program flow with model extended for data assimilation

Simulation Model

Start
↓
Initialize Model
generate mesh
Initialize fields
↓
Do i=1, nsteps
↓
Time stepper
↓
Post-processing
↓
Stop

Assimilation System

Start
↓
init_parallel_pdaf
↓
Initialize Model
generate mesh
Initialize fields
↓
init_pdaf
↓
Do i=1, nsteps
↓
Time stepper
↓
assimilate_pdaf
↓
Post-processing
↓
Stop
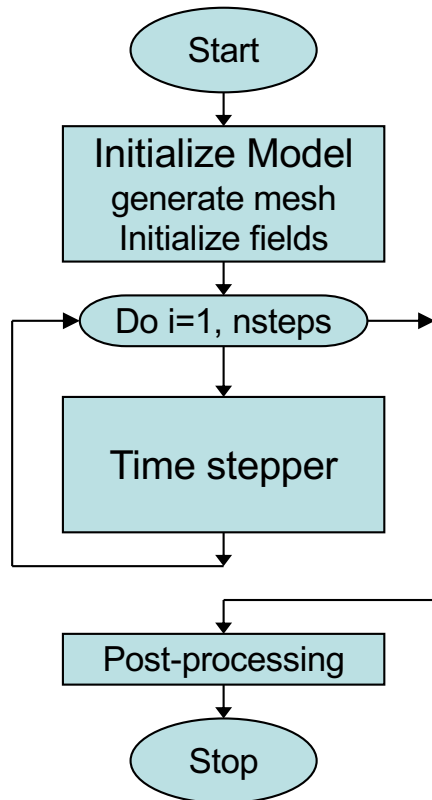
Legend

Model

Extension for data assimilation

PDAF Parallel
Data Assimilation
Framework

Fully parallel configuration

- Tutorial shows implementation for a fully parallel case

  → Number of processes equals ensemble size!

- 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

0e) Inserting subroutine calls

PDAF Parallel Data Assimilation Framework

Where to insert subroutine calls?

`init_parallel_pdaf`

> ➜ at the start of the program
> (first operations to be executed)

`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

- Parallelization variables are declared in Fortran module

  `mod_parallel_pdaf`

- Required adaption:
  - un-comment the second `use model_parallel_pdaf` (includes variables that are declared by the model if it's parallelized)
  - remove local declaration of `mype_world` & `npes_world`

- 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')

PDAF Parallel
Data Assimilation
Framework

init_parallel_pdaf.F90 (2) - Example

The routine initializes 3 groups of communicators

- COMM_model: Used to run the parallel model forecasts
- COMM_filter: Used to compute the filter
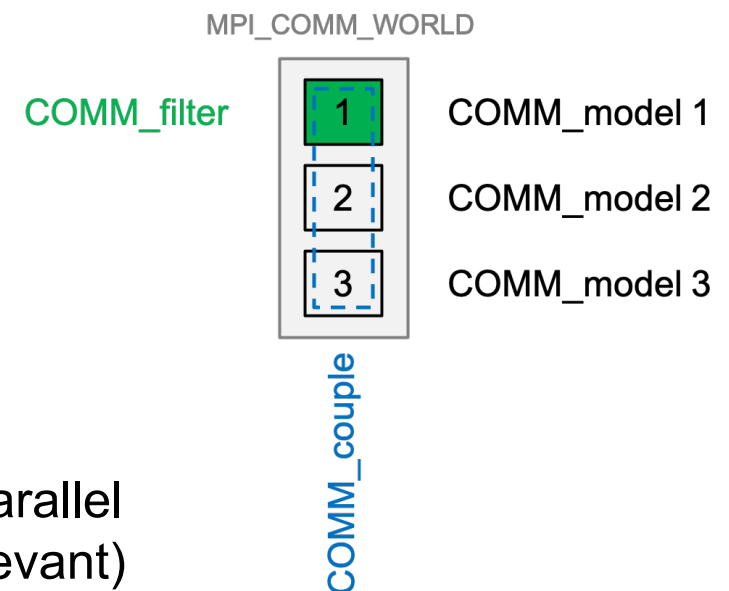- COMM_couple: Coupling between model and filter processes

These are provided to PDAF when calling PDAF_init

The figure shows an example

- 3 processes in total
- 3 model tasks in parallel each using 1 process in its COMM_model
- COMM_couple links the 3 processes to distribute and collect ensemble states
- The filter process uses model task 1

(See also the tutorial for the online mode with a parallel model where the configuration becomes more relevant)

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

MPI_COMM_WORLD

COMM_filter

COMM_couple

1	COMM_model 1
2	COMM_model 2
3	COMM_model 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 state vector

    `dim_state_p = nx * ny`

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

PDAFParallel
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 an interface routine like

```
PDAFomi_assimilate_global
```

We don't insert `PDAFomi_assimilate_global` 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: The template contains calls for all filters

PDAF Parallel
Data Assimilation
Framework

Differences online and offline

- If you've studied the tutorial for offline mode

<table>
<tr><td>

Offline

- Separate programs for model and assimilation

- Needed to implement routine `intialize`

- Grid dimensions declared in `mod_assimilation`

- Ensemble information read from files

- `mod_assimilation` contains all field and assimilation variables

</td><td>

Online

- Extend model program for assimilation

- Operations in `initialize` given by model; no changes for assimilation!

- Grid dimensions defined in model code (`mod_model`)

- Ensemble information provided by model fields

- `mod_assimilation` 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(3)`

        – display timing information
        (setting 1 would give less and 4 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

0f) Forecast phase

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`
`callback_obs_pdafomi.F90` ⎤ analysis step
`obs_A_pdafomi.F90`

`prepoststep_ens_pdaf.F90` ⎤ post step

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`)

Include variables for observation 'A' with
`    USE obs_A_pdafomi, ONLY: assim_A, rms_obs_A`

For the example set :

1. `dim_ens = 9`

2. `rms_obs_A = 0.5`

3. `assim_A = .true.`

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` 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 contents of `field` in column `i` of `ens_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.
(Not relevant until we do parallelization in the analysis step)

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

next_observation_pdaf.F90 (II)

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.

PDAFParallel
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, ny` and `field` with `use mod_model`

2. Initialize model field from state vector:

```
DO j = 1, nx
   field(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/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 9 ./model_pdaf -dim_ens 9`

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

- Outputs are written in `/tutorial/online_2D_serialmodel`

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

  `load state_step10_ana.txt`

  `pcolor(state_step10_ana)`

PDAFParallel
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/online_2D_serialmodel` (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)
  (this is a special result caused by the chosen ensemble. A smaller ensemble (dim_ens 4) leads to much higher difference)

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

PDAF Parallel
Data Assimilation
Framework
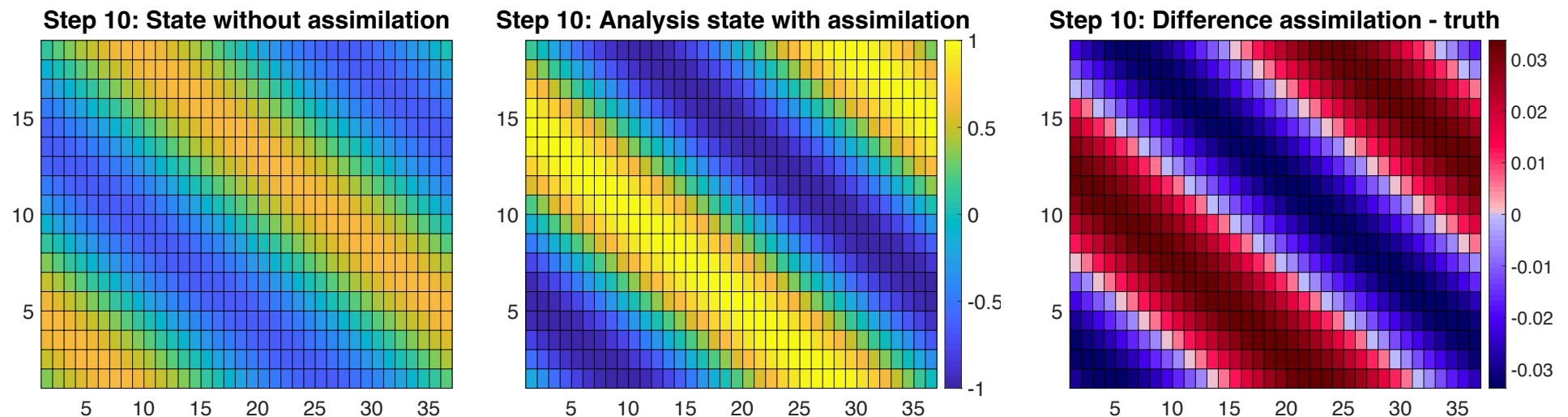
The analysis step

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

PDAFParallel
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
  state_p(1+(j-1)*ny : j*ny) = field(1:ny, j)
END DO
```

Note: The routine independent of the filter!

PDAFParallel
Data Assimilation
Framework

callback_obs_pdafomi.F90

File collecting interface routines for the observation routines called by PDAF

For each observation type we need to add subroutine calls

- Example observation is just called **A**, defined in `obs_A_pdafomi.F90`

In **init_dim_obs_pdafomi**:

- Insert `USE obs_A_pdafomi, ONLY: assim_A, init_dim_obs_A`
- Declare `INTEGER :: dim_obs_A` and set this to zero
- Insert `IF (assim_A) CALL init_dim_obs_A(step, dim_obs_A)`

In **obs_op_pdafomi**:

- Insert `USE obs_A_pdafomi, ONLY: obs_op_A`
- Insert `CALL obs_op_A(dim_p, dim_obs, state_p, ostate)`

(The other observations (B, C) in the file show
how to use multiple observations)

PDAF Parallel
Data Assimilation
Framework

obs_A_pdafomi.F90

PDAF-OMI observation module

- There is a long header with information

Implementation steps from template

- Copy file to name according to observation ('A')

- Replace 'OBSTYPE' by name of observation ('A')

- Implement

    - `init_dim_obs_A`

    - `obs_op_A`

PDAF Parallel
Data Assimilation
Framework

obs_A_pdafomi.F90 (2)

With PDAF-OMI

- Observation Information is stored in Fortran data type `obs_f`

```
TYPE obs_f
   INTEGER :: doassim          ! Whether to assimilate this obs. type
   INTEGER :: disttype         ! Type of distance computation
   INTEGER :: ncoord           ! Number of coordinates
   INTEGER, ALLOCATABLE :: id_obs_p(:,:)
                               ! Indices of observations in state vector

   …
END TYPE obs_f
```

- It is allocated with generic name `thisobs`
  (Motivated by object-oriented programming)

- A single variable, e.g. `disttype`, is accessed in the form

  `thisobs%disttype`

PDAFParallel
Data Assimilation
Framework

init_dim_obs_A in obs_A_pdafomi.F90

Main routine to initialize observation information

- read observation file

- count number of available observations
  (direct output to PDAF: **dim_obs_p**)

- initialize array holding available observations

- initialize array of index of observation in global state vector

- Call `PDAFomi_gather_obs` to finalize initializations

PDAFParallel
Data Assimilation
Framework

init_dim_obs_A in obs_A_pdafomi.F90 (2)

First initializations:

- Specify whether observation is assimilated

  ```
  IF (assim_A) thisobs%doassim = 1
  ```

  (assim_A is included with use and set in init_pdaf)

- Specify type of distance computation (0=Cartesian)

  ```
  thisobs%disttype = 0
  ```

- Number of coordinates used for distance computation

  ```
  thisobs%ncoord = 2
  ```

Note: Parts of the template that are not needed here are deleted from `init_dim_obs_A`

PDAFParallel
Data Assimilation
Framework

Preparations and reading of observation file:

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

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

3. 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 observations (**dim_obs_p**):

1. Declare `integer :: cnt, cnt0`

2. Now count

```
cnt = 0
DO j = 1, nx
  DO i= 1, ny
     IF (obs_field(i,j) > -999.0) cnt = cnt + 1
  END DO
END DO
dim_obs_p = cnt
```

Now we need to initialize

- observation vector   `obs_p`
- inverse variances   `ivar_obs_p`
- index array   `thisobs%id_obs_p`
- observation coordinates   `ocoord_p`

1. All arrays are declared in the template

2. Allocate

   - `obs_p(dim_obs_p)`
   - `ivar_obs_p(dim_obs_p)`
   - `thisobs%id_obs_p(dim_obs_p)`
   - `ocoord_p(2, dim_obs_p)`

3. Initialize these arrays

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

PDAFParallel
Data Assimilation
Framework

3. Now initialize

```
cnt0 = 0                         ! Count grid points
cnt = 0                          ! Count observations
DO j = 1, nx
  DO i= 1, ny
    cnt0 = cnt0 + 1
    IF (obs_field(i,j) > -999.0) THEN
      cnt = cnt + 1
      thisobs%id_obs_p(cnt) = cnt0    ! Index
      obs_p(cnt) = obs_field(i, j)    ! observations
      ocoord_p(1, cnt) = REAL(j)      ! X-coordinates
      ocoord_p(2, cnt) = REAL(i)      ! Y-coordiantes
    END IF
  END DO
END DO
ivar_obs_p(:) = 1.0 / (rms_obs_A*rms_obs_A)
```

PDAF Parallel
Data Assimilation
Framework

obs_op_A in obs_A_pdafomi.F90

Implementation of observation operator
acting one some state vector

Input: state vector `state_p`

Output: observed state vector `ostate`

`init_dim_obs_A` initialized all required information stored in '`thisobs`'

Observation 'A' is defined at grid points

1. Include observation operator routine:

   `USE PDAFomi, ONLY: PDAFomi_obs_op_gridpoint`

2. Call observation operator

   `CALL PDAFomi_obs_op_gridpoint(thisobs, state_p, ostate)`

Note: OMI provides different observation operators,
e.g. for linear interpolation

PDAF Parallel
Data Assimilation
Framework

prepoststep.F90

PDAF-Omi required one small change in prepoststep:

We need to deallocate OMI internal arrays:

For this we insert at the end of the routine

```
CALL deallocate_obs_pdafomi(step)
```

(The routine is included by calback_obs_pdafomi.F90)

Done!

The analysis step in online mode with the serial (non-parallelized) model is fully implemented now

The implementation allows you now to use all global filters! (ESTKF, EKTF, SEIK, EnKF, NETF, PF)

Not usable is SEEK (It's deprecated)

PDAFParallel
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

   - ➡ Tutorial code shows example of 3 observation types

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

   - ➡ Adapt reading/writing routines

PDAFParallel
Data Assimilation
Framework

Differences between online and offline modes

For the analysis step in online mode:

`collect_state_pdaf` - additional routine for online mode

`callback_obs_pdafomi.F90`
                        - identical in online and offline modes

`obs_A_pdafomi.F90:`

`init_dim_obs_A`   - 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

PDAF Parallel
Data Assimilation
Framework

1b) Local filter without parallelization

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
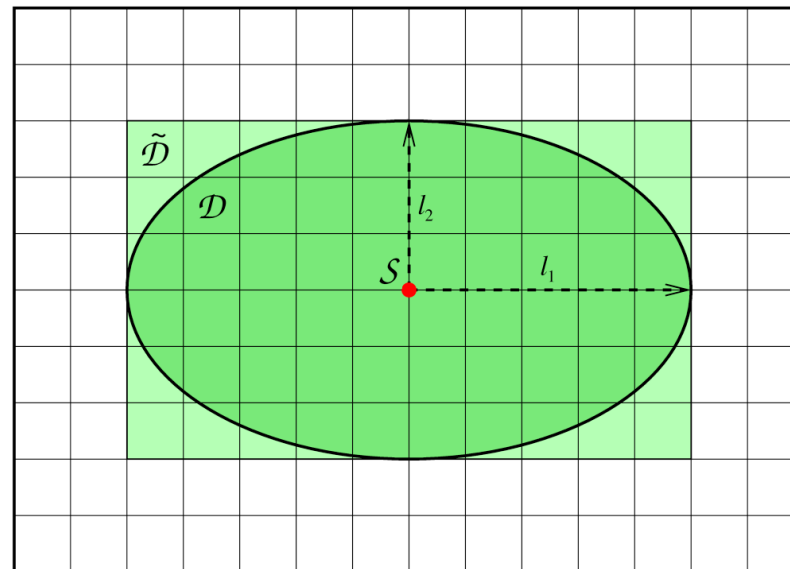
The FULL observation vector

- 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 9 ./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`)

PDAF Parallel
Data Assimilation
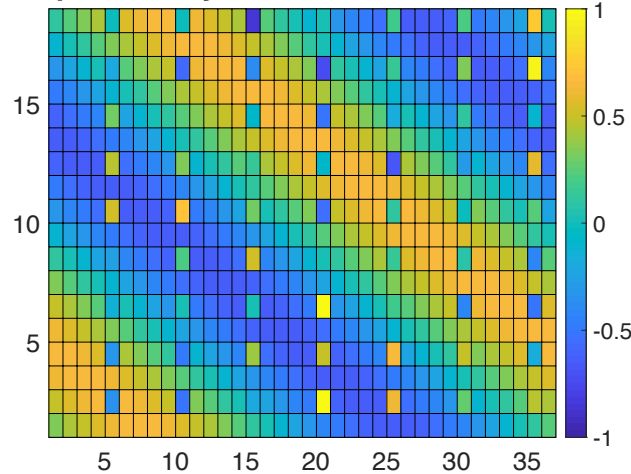Framework

Result of the local assimilation

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

- Default: zero localization radius (cradius=0.0)
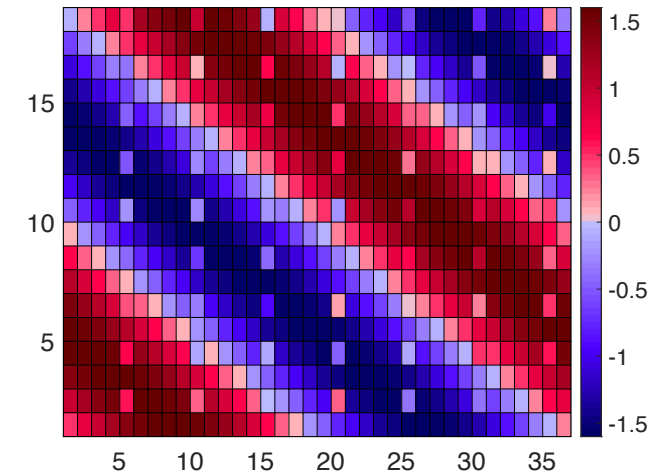
- Change only at observation locations

Step 10: State without assimilation
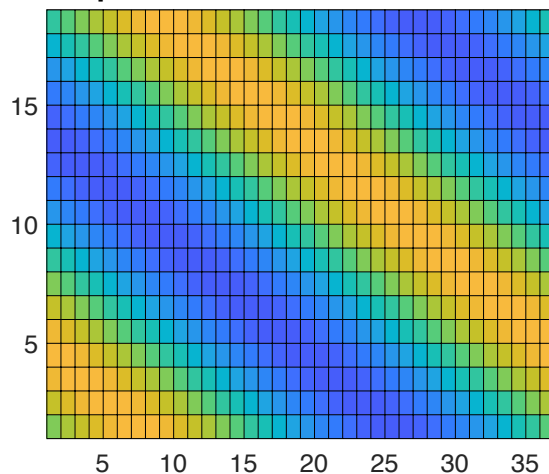
Step 10: Analysis state from assimilation
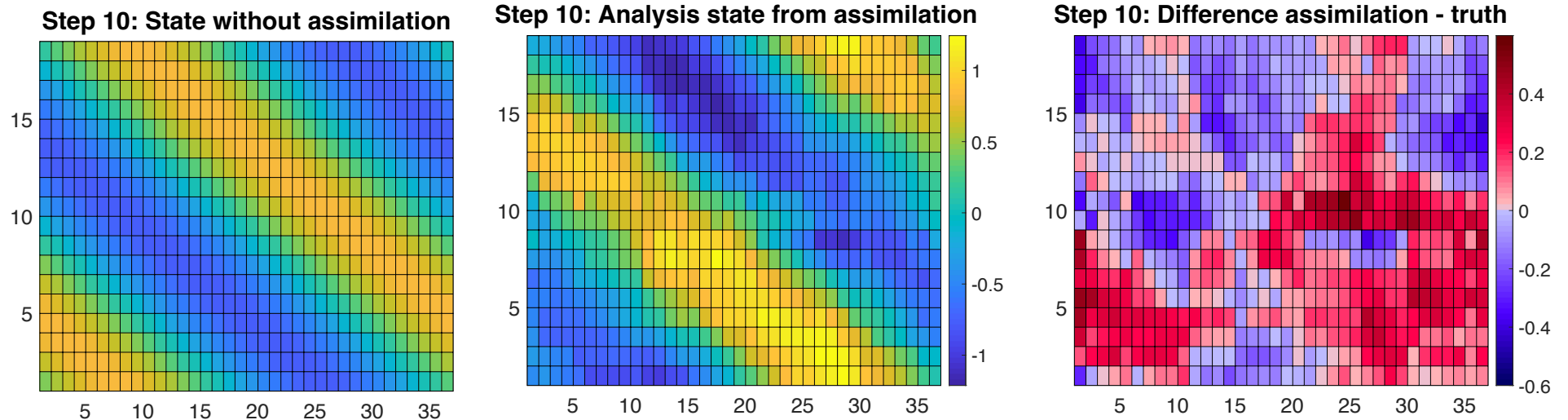
Step 10: Difference assimilation - truth

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

Step 10: State without assimilation

Step 10: Analysis state from assimilation

Step 10: Difference assimilation - truth

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)

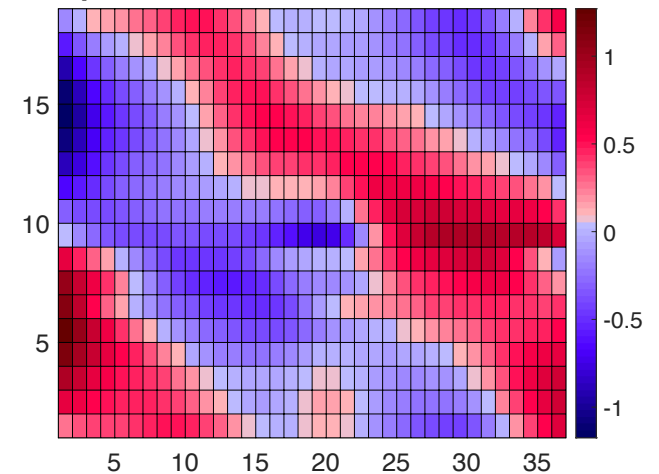- The high errors on the left are due to the distance of observations

**Step 10: State without assimilation** **Step 10: Analysis state from assimilation** **Step 10: Difference assimilation - truth**
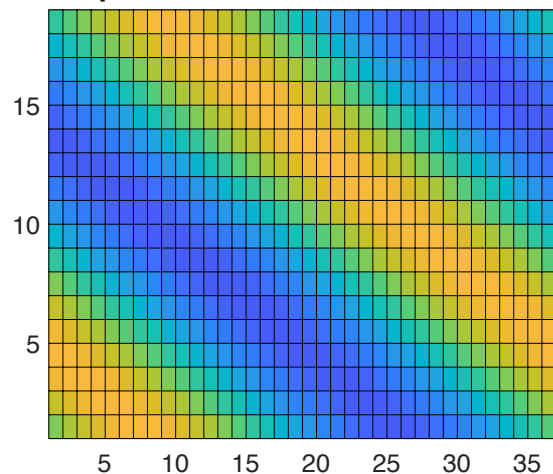
PDAF Parallel
Data Assimilation
Framework
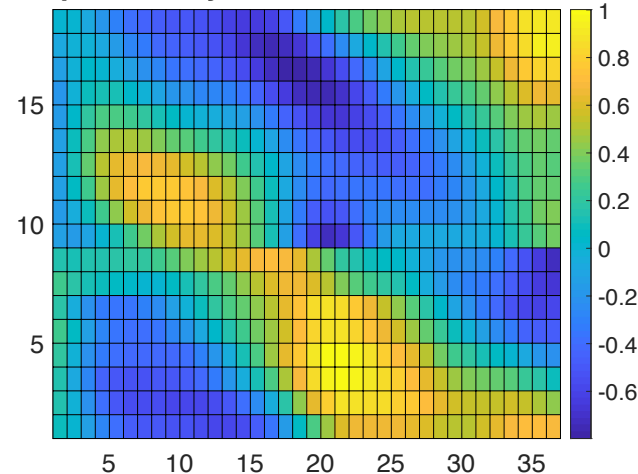
Result of the local assimilation (3)

`… -filtertype 7 -cradius 40.0`

- Large radius: All local analysis domains see all observations

- Result identical to global filter

Step 10: State without assimilation   Step 10: Analysis state from assimilation   Step 10: Difference assimilation - truth

PDAF Parallel
Data Assimilation
Framework
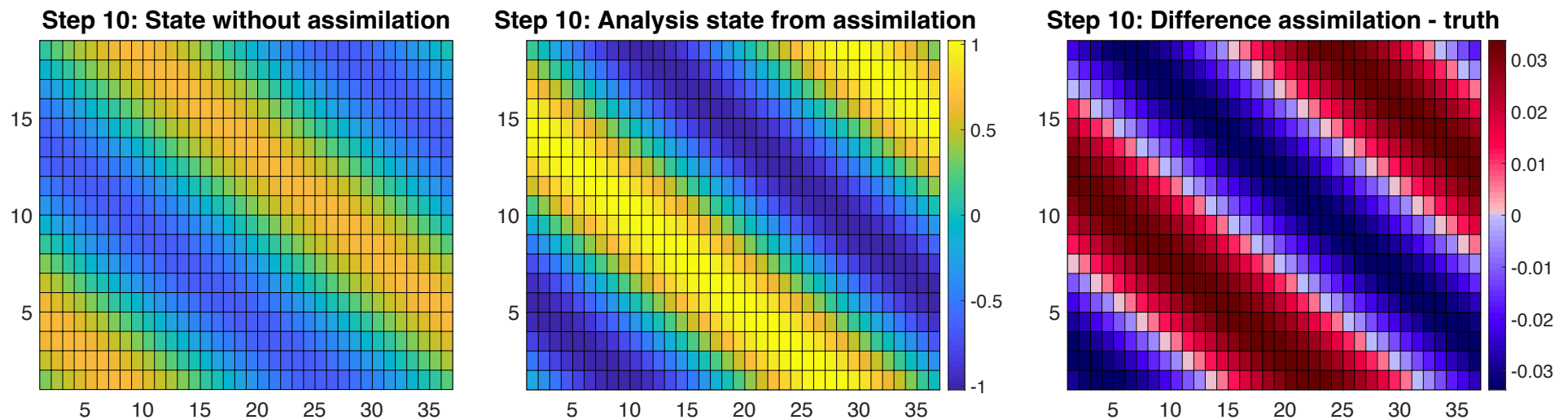
Local filter LESTKF

- 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)

Additional files for local analysis step

init_n_domains_pdaf.F90

init_dim_l_pdaf.F90

g2l_state_pdaf.F90

l2g_state_pdaf.F90

localize
state vector

Additional routine in callback_obs_pdafomi.F90:

init_dim_obs_l_pdafomi

localize
observations

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 grid points (nx * ny)

To do:

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

    n_domains_p = nx * ny

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

2. Compute the coordinates:

    - Include `coords_l` with `use mod_assimilation`

    `coords_l(1) = REAL(CEILING(REAL(domain_p)/REAL(ny)))`

    `coords_l(2) = REAL(domain_p) - (coords_l(1)-1)*REAL(ny)`

Note: `coords_l` will be used later for computing the distance of observations form the local analysis domain in `init_dim_l_pdaf`

PDAF Parallel
Data Assimilation
Framework

init_dim_l_pdaf.F90 (2)

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

callback_obs_pdafomi.F90

File collecting interface routines for the observation routines called by PDAF

For each observation type we need to add subroutine calls

- The example observation is just called **A**, defined in `obs_A_pdafomi.F90`

In **init_dim_obs_l_pdafomi**:

- Insert

  `USE obs_A_pdafomi, ONLY: init_dim_obs_l_A`

- Insert

  `CALL init_dim_obs_l_A(domain_p, step, dim_obs, dim_obs_l)`

(The other observations (B, C) in the file show
how to use multiple observations)

PDAF Parallel
Data Assimilation
Framework

init_dim_obs_l_pdaf.F90

Set size of the observation vector for the local analysis domain and initialize local observation information

Only direct output: **`dim_obs_l`**

Operations:

1. With `use mod_assimilation`

   - Include coordinates `coords_l`

   - Include localization variables (`cradius, locweight, sradius`)

2. Call `PDAFomi_init_dim_obs_l` to perform necessary operations

Note: we use a fixed radius cradius here. One could make it varying with the local analysis domain. Also it could vary with observation type.

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!

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 all local filters! (LESTKF, LETKF, LSEIK, LNETF)

Not usable is LEnKF
(It needs one more routine (localize_covariance_pdafomi) which we don't discuss here; but it's coded in the tutorial code)

For testing one can vary localization parameters:

`cradius` – the localization 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

A complete local analysis step

We now have a fully functional analysis step including localization

➢ It can be adapted to multiple model fields, 3 dimensions, different observations, etc.

➢ It can be used even with big models

- if computing time is no concern

- and if the computer has sufficient memory (e.g. ensemble array with dimension 10^7 and 20 members requires about 1.6 GB)

➢ Parallelization of the analysis step

- is required if the problem is too big for a single process

- is recommended if you used a parallelized model

PDAFParallel
Data Assimilation
Framework

1b.1) Use local filter with OpenMP-parallelization

PDAF Parallel
Data Assimilation
Framework

OpenMP

- OpenMP is so-called *shared-memory parallelization*

- Support for OpenMP is built into current compilers
  (needs to be activated by compiler-flag)

- Define OpenMP in the code by compiler directives: `!$OMP` ...

- Shared-memory parallelization:

  - Run several OpenMP "threads" (like processes in MPI)

  - All threads can access the same array in memory, but perform different operations

  - Typical is loop-parallelization: Each thread executes some part of a loop. For example, a fraction of a vector:

```
!$OMP parallel do
DO i = 1, 1000
    a(i) = b(i) + c(i)
ENDDO
```

With 2 threads, typically:
- thread 1 runs i=1 to 500
- thread 2 runs i=501 to 1000

PDAF Parallel
Data Assimilation
Framework

OpenMP – what's relevant for PDAF

The local filters (LESTKF, LETKF, LSEIK, LNETF) are parallelized with OpenMP

- ➢ The loop over local analysis domains is distributed over threads

To make this work:

- ➢ Take into account, whether a variable is

  - • *shared* (all treads see the same) or

  - • *private* (each thread has it's own copy)

- ➢ Variables referring to a local analysis domain (e.g. coords_l) have to be private

- ➢ This is ensured using the declaration 'THREADPRIVATE'

PDAF Parallel
Data Assimilation
Framework

Running the tutorial program

Run analogously to case without parallelization

- cd to `/tutorial/online_2D_serialmodel`

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

- Check and edit the make include file to activate OpenMP

    - for gfortran: `OPT = … -fopenmp`

    - for Intel compiler: `OPT = … -openmp`

- Compile by running `'make'`

- Set the number of OpenMP threads as environment variable, e.g.

    - for bash: `export OMP_NUM_THREADS=2`

    - for tcsh: `setenv OMP_NUM_THREADS 2`

- Run the program as without OpenMP-parallelization

PDAF Parallel
Data Assimilation
Framework

Results from running with OpenMP parallelization

The results should be *identical* to those without parallelization

If the program is compiled with activated OpenMP-parallelization, you will see in the output of the analysis step the line

```
--- Use OpenMP parallelization with    2 threads
```

PDAF Parallel
Data Assimilation
Framework

OpenMP in the local filters

PDAF supports the use of OpenMP in the localized filters (LESTKF, LETKF, LSEIK, LNETF)

Settings to make OpenMP work are in: `mod_assimilation.F90`
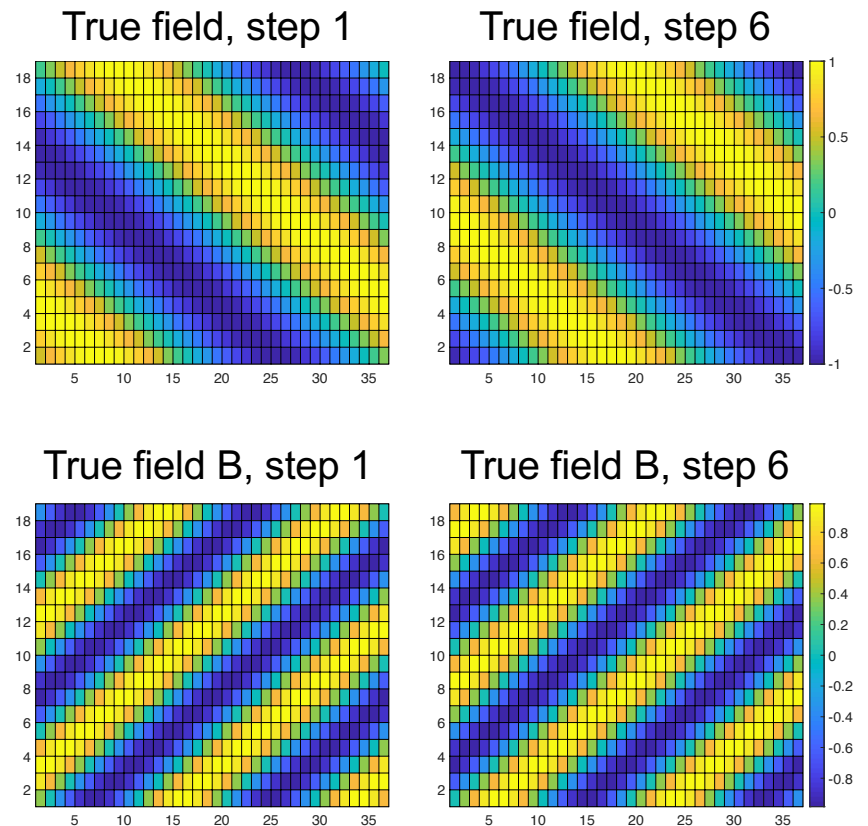
Last line of `mod_assimilation.F90` is

`!$OMP THREADPRIVATE(coords_l, id_lstate_in_pstate)`

➢ These variables are specific for each local analysis domain

➢ The variables are declared in mod_assimilation.F90

➢ The declaration 'THREADPRIVATE' ensures that each variable can have a different value in the different threads

PDAF Parallel
Data Assimilation
Framework

2) Adaption to model with 2-fields

PDAF Parallel
Data Assimilation
Framework

2D „Model" with 2 fields

- See the separate tutorial slides about the models

- The 2-field model example is in `/tutorial/ online_2D_serialmodel_2fields`

- Simple 2-dimensional grid domain

- 36 x 18 grid points (longitude x latitude)

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

True field, step 1

True field, step 6

True field B, step 1

True field B, step 6

PDAF Parallel
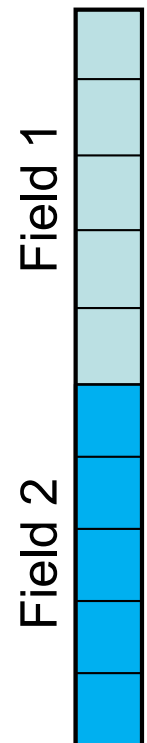**Data Assimilation**
Framework

Adaptation for 2 fields in state vector

Here we show an efficient strategy to handle the 2 fields
based on what we also implement with complex models

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

- For two or more fields:

  - concatenate them in the state vector

  - Adapt fields definition in `mod_assimilation.F90`

  - adapt state dimension in `init_pdaf`

  - Add arrays for field offsets and dimensions in `init_pdaf`

  - adapt `init_ens_pdaf, collect_state_pdaf, distribute_state_pdaf, prepoststep_pdaf`

  - For local filters: Adapt `init_dim_l_pdaf`

  - Adapt observation modules
    (in particular to ensure that the observation operator
    uses the correct field)

State vector
with 2 fields

Field 1

Field 2

PDAF Parallel
Data Assimilation
Framework

mod_assimilation.F90

In template of `mod_assimilation` we have declared

```
INTEGER :: n_fields        ! number of fields in state vector
INTEGER, ALLOCATABLE :: off_fields(:)  ! Offsets of fields in state vector
INTEGER, ALLOCATABLE :: dim_fields(:)  ! Dimension of fields in state vector

TYPE field_ids               ! Derived type for indexing
      INTEGER :: NAME_OF_FIELD_1
!     INTEGER :: NAME_OF_FIELD_2
!     INTEGER :: ...
END TYPE field_ids
TYPE(field_ids) :: id       ! Declare 'id' to be of type(field_ids)
```

- TYPE 'field_ids' is used to define index of a field in the state vector
  - allows to give each field a name
  - Adapt for variables included in an application

  For 2 fields called 'fieldA' and 'fieldB':
```
      TYPE field_ids
         INTEGER :: fieldA
         INTEGER :: fieldB
      END TYPE field_ids
```

PDAF Parallel
Data Assimilation
Framework

init_pdaf.F90

1. Adapt dimension of state vector (now 2 fields):

```
dim_state_p = 2 * nx * ny
```

2. Add vectors for field dimensions and offsets in state vector:

In `init_pdaf` use-include these variables and initialize

```
n_fields = 2                          ! Number of model fields

id%fieldA = 1                         ! Index of fieldA
id%fieldB = 2                         ! Index of fieldB
allocate(off_fields(n_fields))
allocate(dim_fields(n_fields))

dim_fields(1) = nx * ny      ! Dimension of fieldA
dim_fields(2) = nx * ny      ! Dimension of fieldB

off_fields(1) = 0                     ! Offsets start with 0 (field starts at 1)
DO i = 2, n_fields                    ! Generic; this works with any number of fields
   off_fields(i) = off_fields(i-1) + dim_fields(i-1)
END DO
```

- These arrays allow for a flexible code in other routines
- With parallelization one usually needs to distinguish global 'dim_fields' and process-local 'dim_fields_p'

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

PDAF Parallel
Data Assimilation
Framework

init_ens_pdaf.F90

Initialize ensemble matrix `ens_p` with both fields

1. Include `off_fields` with `use mod_assimilation`

2. For **first field**

   - modify the initialization of `ens_p` to account for `off_fields`:

```
DO j = 1, nx
 ens_p(off_fields(id%fieldA) + 1+(j-1)*ny : &
    off_fields(id%fieldA) + j*ny, member) = readfield(1:ny, j)
END DO
```

3. For **second field** (fieldB)

   - copy the lines reading the first field

   - Adapt file reading to read from ensB_*

   - Fill `ens_p` with offset `off_fields(id%fieldB)`:

```
DO j = 1, nx
 ens_p(off_fields(id%fieldB) + 1+(j-1)*ny : &
    off_fields(id%fieldB) + j*ny, member) = readfield(1:ny, j)
END DO
```

Note: This is independent of the actual values in `off_fields`

PDAF Parallel Data Assimilation Framework

distribute_state_pdaf.F90 / collect_state_pdaf.F90

distribute_state_pdaf:
Write both fields from state vector into field arrays:

1. Include `fieldB` with `use mod_model`

2. Include `off_fields` with `use mod_assimilation`

3. Write information from state vector into both fields utilizing `off_fields` and `id`:

```
DO j = 1, nx
  field(1:ny, j) = state_p(off_fields(id%fieldA)+1+(j-1)*ny : &
          off_fields(id%fieldA)+j*ny)
END DO
DO j = 1, nx
  fieldB(1:ny, j) = state_p(off_fields(id%fieldB)+1+(j-1)*ny :&
          off_fields(id%fieldB)+j*ny)
END DO
```

collect_state_pdaf:
Same operations but in reversed order

PDAF Parallel
Data Assimilation
Framework

prepoststep_ens_pdaf.F90

Changes for 2 fields:

1. Compute RMS errors separately for each field
2. Write `field` and `fieldB` into separate files (copy outputs and adapt)

1. Computing RMS error:

- Include `n_fields, dim_fields, off_fields` with `use mod_assimilation`

- Change `rmserrors_est` into an allocatable array and allocate with dimension `n_fields`

- Compute separate values of rmserrors_est:

```
DO j = 1, n_fields
   DO i = 1+off_fields(j), dim_fields(j)+off_fields(j)
      rmserror_est(j) = rmserror_est(j) + variance(i)
   ENDDO
   rmserror_est(j) = SQRT(rmserror_est(j) / dim_fields(j))
ENDDO
```

Note: This form is generic and works for any number of fields

PDAF Parallel
Data Assimilation
Framework

obs_A_pdafomi.F90

PDAF-OMI observation module

- **observations of 'field'**

- Changes to account for existence of two fields:

  - need to account for the position of the observed field in the state vector, described by `off_field`

  - Include `off_fields` with `use mod_assimilation`

  - Account for `off_fields` when initializing `this%id_obs_p`:

```
thisobs%id_obs_p(1, cnt) = cnt0 + off_fields(id%fieldA)
```

- In addition we change the file reading to read from the directory `tutorial/inputs_offline_2fields/`

PDAF Parallel
Data Assimilation
Framework

obs_B_pdafomi.F90

PDAF-OMI observation module

- **observations of 'fieldB'**

- Changes to account for existence of two fields:

  - need to account for the position of the observed field in the state vector, described by `off_field`

  - Include `off_fields` with `use mod_assimilation`

  - Account for `off_fields` when initializing `this%id_obs_p`:

```
thisobs%id_obs_p(1, cnt) = cnt0 + off_fields(id%fieldB)
```

  - In addition we change the file reading to read from the directory `tutorial/inputs_offline_2fields/`

PDAF Parallel
Data Assimilation
Framework

init_dim_l_pdaf.F90

Local state vector now contains 2 fields (or `n_fields` in general):

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

1. Include `n_fields, off_fields`
    with `use mod_assimilation`

2. Set `dim_l = n_fields`

3. Revise initialization of `id_lstate_in_pstate`:

```
DO i=1, n_fields
    id_lstate_in_pstate(i) = domain_p + off_fields(i)
END DO
```

Note:

*   This is generic, but assumes that each field has only one element in local state vector

PDAF Parallel
Data Assimilation
Framework

A complete analysis step for global or local filters

The adaption for 2 fields is now complete

➢ One can assimilate observation types A and B either separately or jointly. Possible command line options:

```
-assim_A .true. -assim_B .false.      (default)

-assim_A .false. -assim_B .true.

-assim_A .true. -assim_B .true.
```

➢ The changes for 2 fields are analogous for a parallelized model (one has to account for the situation that the global state vector contains both fields, while the state vector on each process contains two sub-fields)

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 in `init_pdaf`

  - Adapt `distribute_state` and `collect_state`

  - Adapt `init_dim_l`

  - Adapt observation modules and `callback_obs_pdafomi`

- 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)

PDAF Parallel
Data Assimilation
Framework

Multiple fields in state vector

- Tutorial uses a single 2-dimensional field or two 2D fields

    - See section 3 for 2-fields case

- Adaptation to mixed 2D and 3D fields needs care if local analysis domain is more than single grid point (like a vertical column in a 3D model)

State vector with 2 fields

Field 1

Field 2

PDAF Parallel
Data Assimilation
Framework
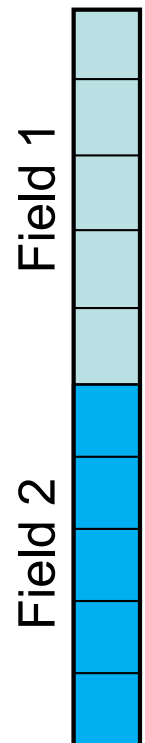
Multiple observed fields

- In tutorial

  - We discussed observations of a field at some grid points (two different observed fields in 2-fields case, section 2)

  - Example code shows three different observation types

- For several observed fields adapt observation routines:

  - Create a new observation module (`obs_OBSTYPE_pdafomi.F90`)

  - Add calls to routine in `callback_obs_pdafomi.F90`

- **Note**

  - The observation errors can be set differently for each observed field

  - The localization radius can be set specific for each observed field (use a different variable `cradius_OBSTYPE`)

PDAF Parallel
Data Assimilation
Framework

The End!

Tutorial described example implementations

- Online mode of PDAF parallelized over ensemble members

- Simple 2D model without parallelization and with OpenMP parallelization

- Implementation supports various filters

  - 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

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