PDAF Tutorial

Implementation of the analysis step in offline mode using PDAF-OMI

http://pdaf.awi.de

PDAF Parallel Data Assimilation Framework

V1.13 – 2024-08-25

Implementation Tutorial for PDAF offline

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 V2.3 and later)

PDAF Parallel Data Assimilation Framework

Implementation Tutorial for PDAF offline

This is just an example!

For the complete documentation of PDAF's interface

see the documentation

at http://pdaf.awi.de

Overview

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

4 Parts

1. Without parallelization
   a) Global filter
   b) Localized filter
   (and OpenMP-parallelization)
2. With MPI-parallelization
   a) Global filter
   b) Localized filter

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

We assume that 1a is implemented before 1b and 1a is implemented before 2a (1b before 2b).

Contents

0a) Files for the Tutorial

PDAF Parallel Data Assimilation Framework

Tutorial implementation

Files are in the PDAF package

Directories:

`/tutorial/offline_2D_serial`	(only OpenMP-parallelization)
`/tutorial/offline_2D_parallel`	(with MPI parallelization)

- 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

Templates for offline mode

Directory: `/templates/offline_omi`

- Contains all required files
- Contains also
  command line parser, memory counting, timers
  (convenient but not required)

To generate your own implementation:

1. Copy directory to a new name
2. Complete routines for your model
3. Set base directory (`BASEDIR`) in Makefile
4. Set `$PDAF_ARCH`
5. Compile

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 run ‘make’ in the main directory of PDAF (requires setting of PDAF_ARCH)

`$PDAF_ARCH`

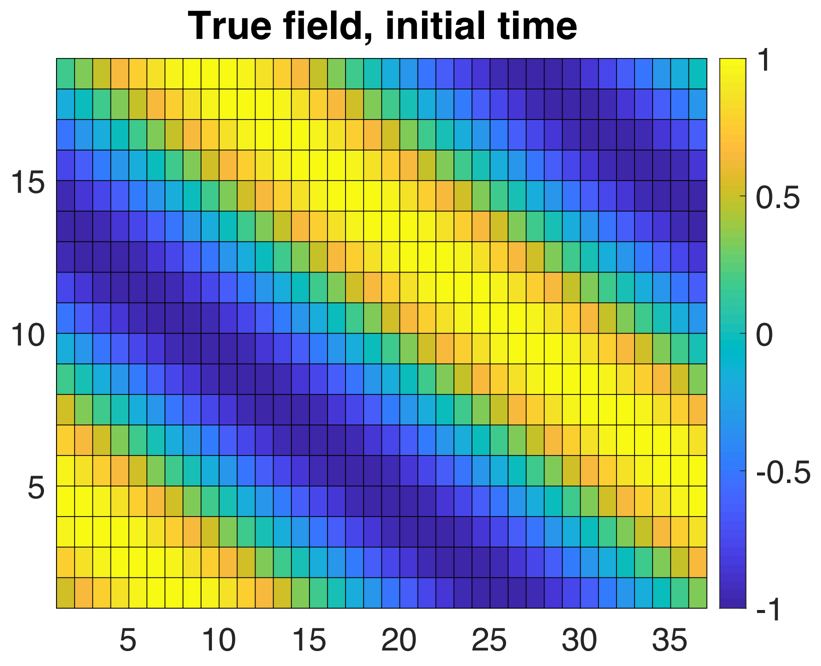
- Environment variable to specify the compile specifications
- Definition files in `/make.arch`
- Define by, e.g.
  `setenv PDAF_ARCH linux_gfortran` (tcsh/csh)
  `export PDAF_ARCH=linux_gfortran` (bash)

PDAF Parallel Data Assimilation Framework

0b) The 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
- No dynamics for offline mode
- Stored in text file (18 rows) – `true.txt`



PDAF Parallel 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

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

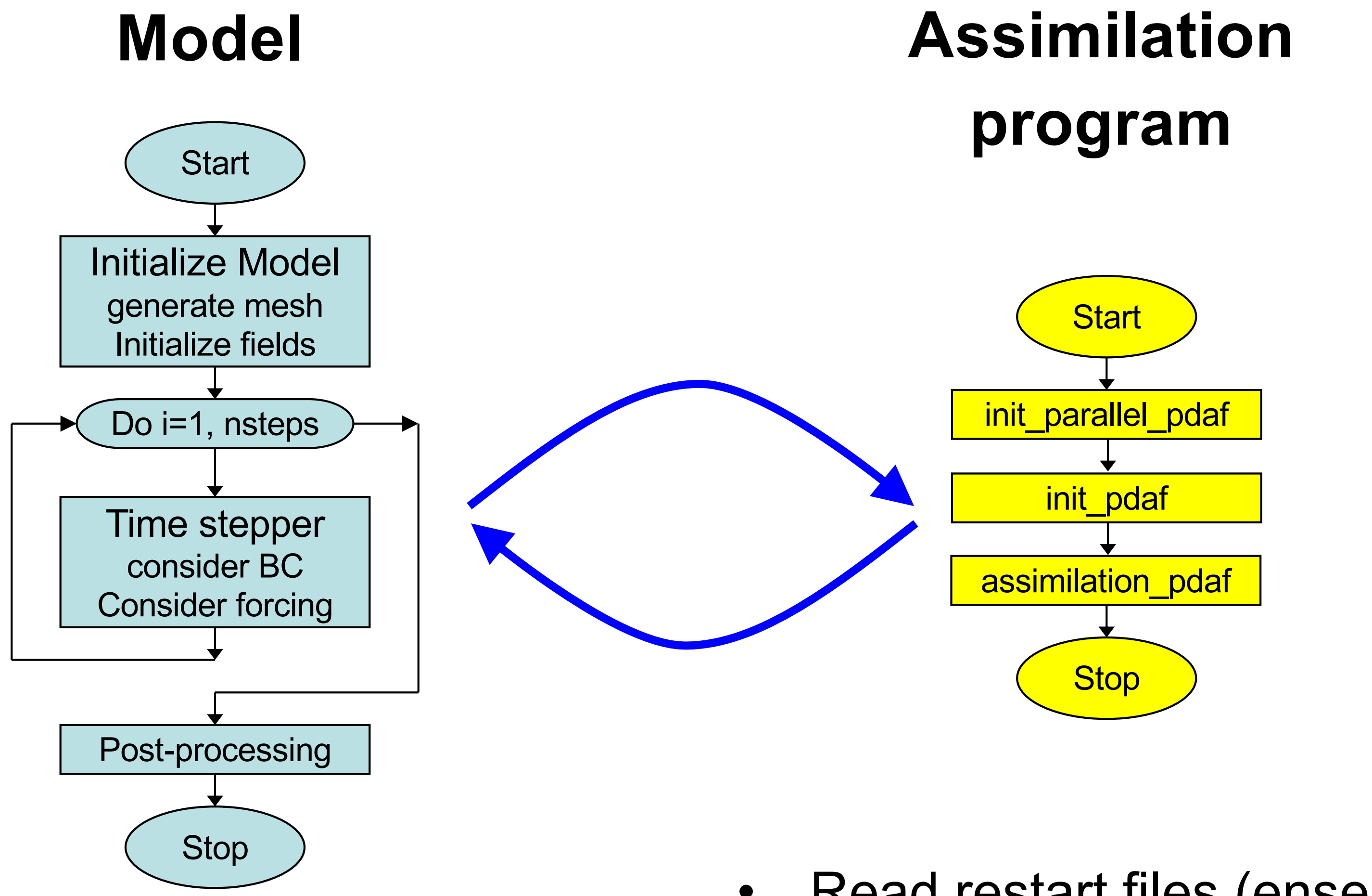
0d) PDAF offline mode

Offline mode

- Two separate programs
  - `“Model”` – performs ensemble integrations
  - `“PDAF_offline”` – perform analysis step

- Couple both programs through files
  1. `“PDAF_offline”` reads ensemble forecast files
  2. Performs analysis step
  3. Writes analysis ensemble files (restart files for “Model”)
  4. `“Model”` reads restart files and performs ensemble integration

Programs in offline mode



- Run for each ensemble member
- Write restart files

- Read restart files (ensemble)
- Compute analysis step
- Write new restart files

PDAF Parallel Data Assimilation Framework

`PDAF_offline`: General program structure

```
program main_offline

   init_parallel_pdaf
                              initialize communicators
                              (not relevant without parallelization)

   initialize
                              initialize model information

   init_pdaf
                              initialize parameters for PDAF
                              and read ensemble

   assimilation_pdaf
                              perform analysis
                              (by call to PDAF_put_state_X)

end program
```

PDAF Parallel Data Assimilation Framework

1 Filters without parallelization

1a) Global filter without parallelization

Running the tutorial program

- Do `cd /tutorial/offline_2D_serial`
- Set environment variable `PDAF_ARCH` or specify it when running make (e.g. `linux_gfortran`)
- Compile by running `'make'` (or `'make PDAF_ARCH=…'`) (next slide will discuss possible compile issues)
- Run the program with `./PDAF_offline`

- Inputs are read in from `/tutorial/inputs_offline`
- Outputs are written in `/tutorial/offline_2D_serial`
- Plot result, e.g. with Python:

```
python ../plotting/plot_file.py state_ana.txt
```

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` 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)

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`

Files in the tutorial implementation

`/tutorial/inputs_offline`

• `true.txt`	true state
• `state_ini.txt`	initial estimate (ensemble mean)
• `obs.txt`	observations
• `ens_X.txt` (X=1,…, 9)	ensemble members

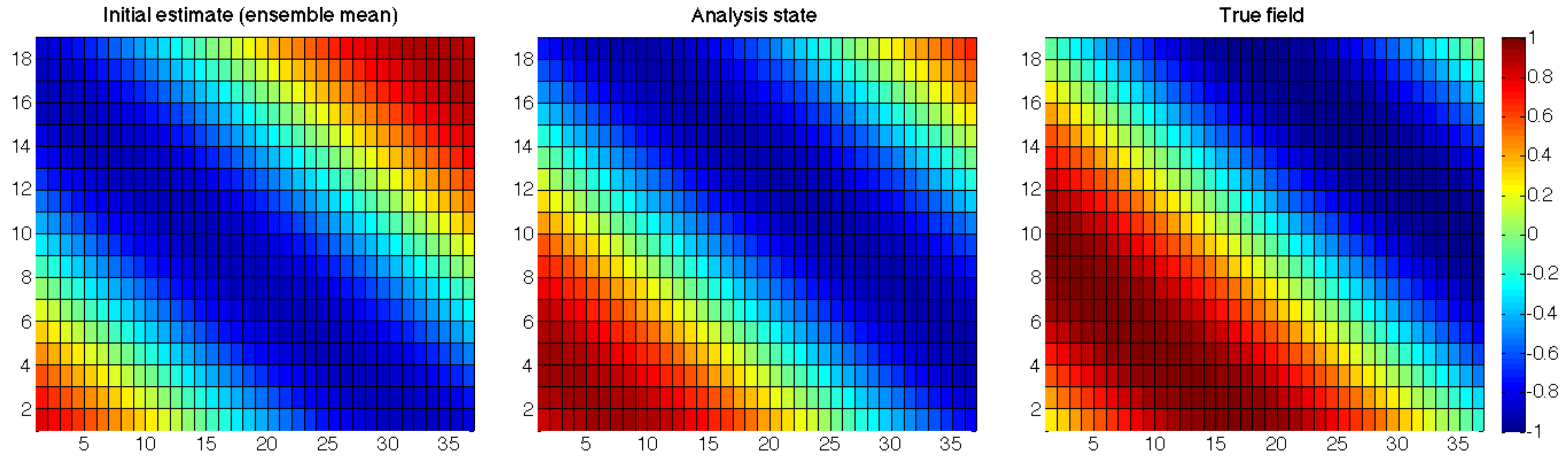
`/tutorial/offline_2D_serial` (after running PDAF_offline)

• `state_ana.txt`	analysis state estimate
• `ens_X_ana.txt` (X=1,…,9)	analysis ensemble members

PDAF Parallel Data Assimilation Framework

Result of the global assimilation

- The analysis state is closer to the true field than the initial estimate
- Truth and analysis are not identical
  (the ensemble does not allow it)



PDAF tutorial – Analysis step in offline mode

Files for PDAF

Template contains all required files

➢ just need to be filled with functionality

`mod_assimilation_offline.F90`	Fortran module

`initialize.F90`
`init_pdaf_offline.F90`
`init_ens_offline.F90`
	initialization

`callback_obs_pdafomi.F90` `obs_A_pdafomi.F90`	analysis step
`prepoststep_ens_offline.F90`	post step

mod_assimilation.F90

Fortran module

- Declares the parameters used to configure PDAF
- Add model-specific variables here
  (see next slides)
- Will be included (with ‘`use`’) in the user-written routines

PDAF Parallel Data Assimilation Framework

initialize.F90

Routine initializes the model information

1. Define 2D mesh in mod_assimilation.F90

   ```
   integer :: nx, ny
   ```

2. In initialize.F90 include `nx, ny,` and `dim_state_p` with `use mod_assimilation`

3. Define mesh size in initialize.F90

   ```
   nx = 36
   ny = 18
   ```

4. Define state dimension in initialize.F90

   ```
   dim_state_p = nx * ny
   ```

Note: Some variables end with `_p`.
It means that the variable is specific for a process.
(Not relevant until we do parallelization)

init_pdaf_offline.F90

Routine sets parameters for PDAF, calls `PDAF_init` to initialize the data assimilation, and `PDAF_set_offline_mode` to activate the offline mode:

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.`
4. `filtertype = 6` (for ESTKF)

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

```
init_ens_offline
```

init_ens_offline.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_offline(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.

init_ens_offline.F90 (2)

Initialize ensemble matrix `ens_p`

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

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

The analysis step

At this point the initialization of PDAF is complete:

- Forecast ensemble is initialized
- Filter algorithm and its parameters are chosen

Next:

- Implement user-routines for analysis step
- 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)

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

- 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)

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 ‘TYPE’ 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`

PDAF Parallel 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

PDAF Parallel 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 in `init_dim_obs_A`

PDAF Parallel Data Assimilation Framework

init_dim_obs_A in obs_A_pdafomi.F90 (3)

Preparations and reading of observation file:

1. Include `nx, ny` with `use mod_assimilation`
2. declare and allocate real array `obs_field(ny, nx)`
3. read observation file:

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

PDAF Parallel Data Assimilation Framework

init_dim_obs_A in obs_A_pdafomi.F90 (4)

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
```

init_dim_obs_A in obs_A_pdafomi.F90 (5)

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.

PDAF Parallel Data Assimilation Framework

init_dim_obs_A in obs_A_pdafomi.F90 (6)

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_ens_offline.F90

Post-step routine for the offline 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`

Required extension:

4. Write analysis ensemble into files used for model restart (Analogous to reading in `init_ens_offline`)

Possible (useful) extension:

5. Write analysis state (ensemble mean, `state_ana.txt`)

PDAF Parallel Data Assimilation Framework

Done!

The analysis step in offline mode 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)

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

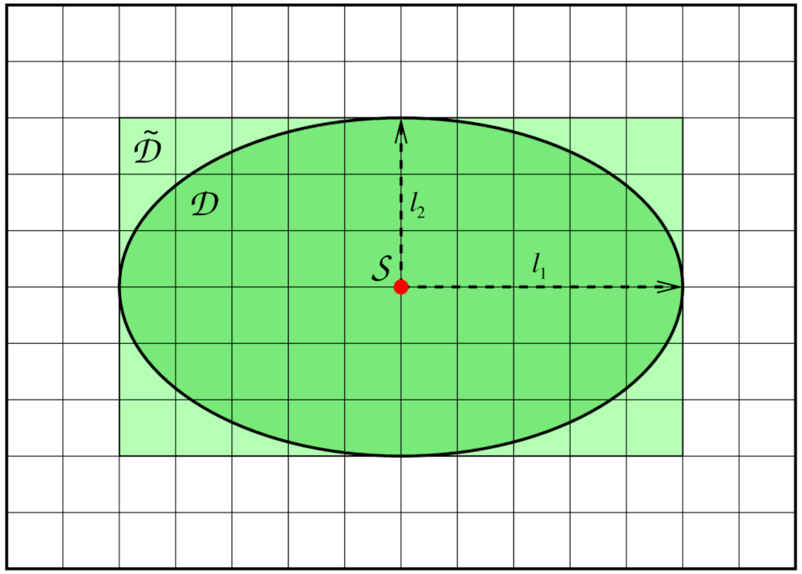
PDAF Parallel Data Assimilation Framework

1b) Local filter without parallelization

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*),
  e.g. defined by the ellipse or rectangle
- Weigh 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 local 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

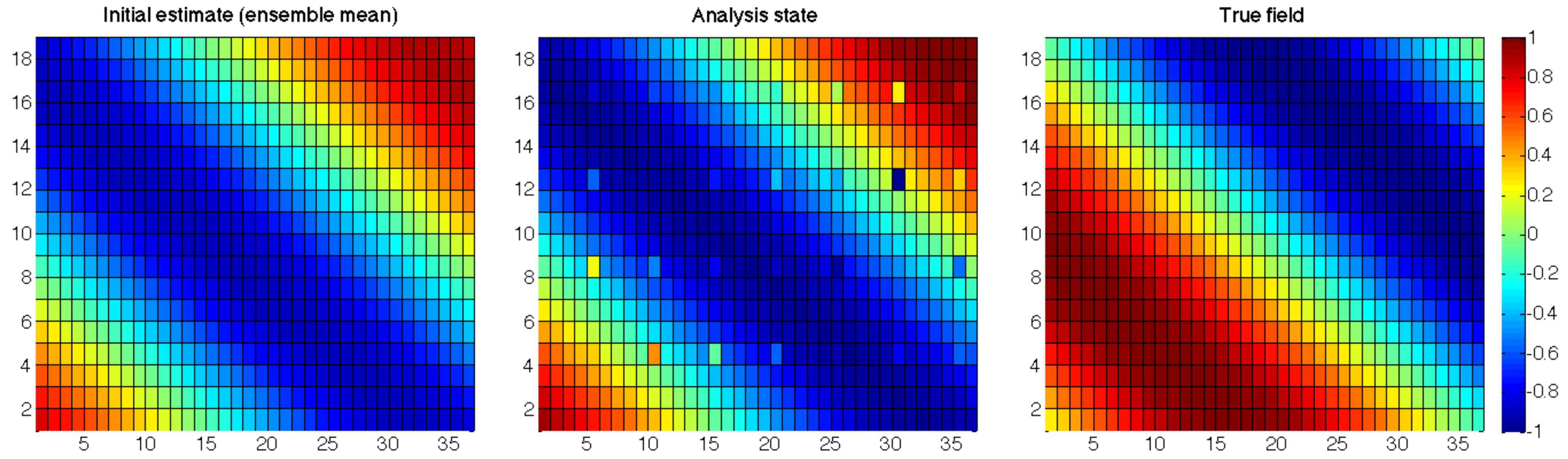
Running the tutorial program

- Compile as for the global filter
- Run the program with `./PDAF_offline OPTIONS`
- `OPTIONS` are always of type `-KEYWORD VALUE`
- Possible `OPTIONS` are
  - `-filtertype 7` (select LESTKF if not set in init_pdaf_offline)
  - `-cradius 5.0` (set localization cut-off 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_offline`)

Result of the local assimilation

```
./PDAF_offline -filtertype 7
```

- Default: zero localization radius (cradius=0.0)
- State is changed only at observation locations



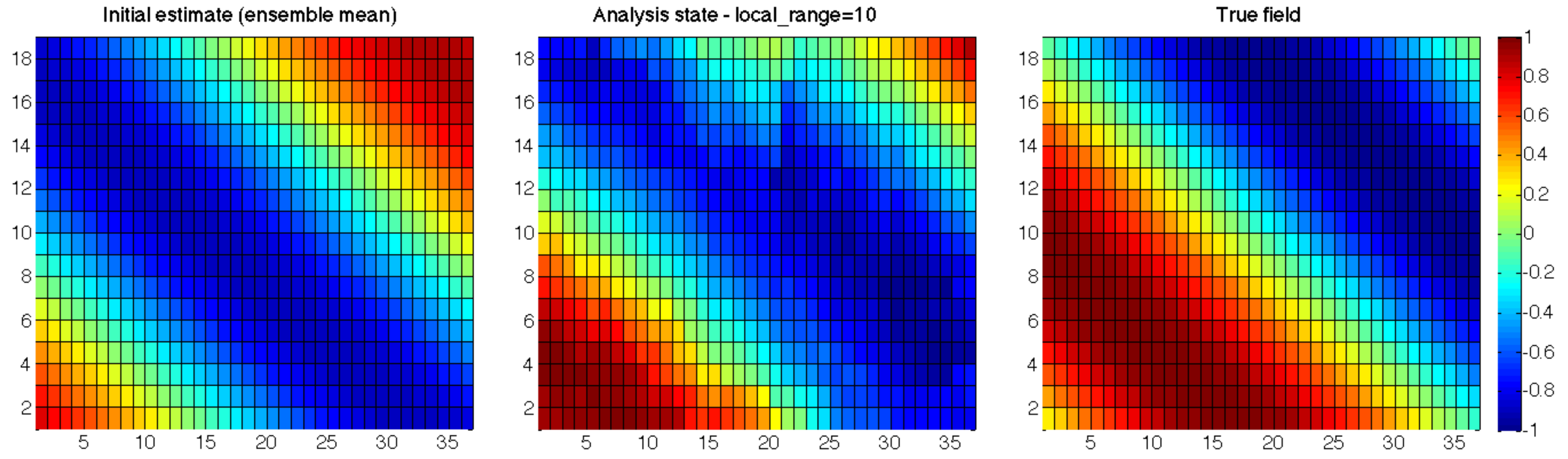
PDAF Parallel Data Assimilation Framework

Result of the local assimilation (2)

```
./PDAF_offline -filtertype 7 -cradius 10.0
```

- All local analysis domains are influenced (all see observations)
- Up to 16 observations used in a single local analysis (average 9.6)

Note: The the shape of the ensemble members favors the global filter in this experiment

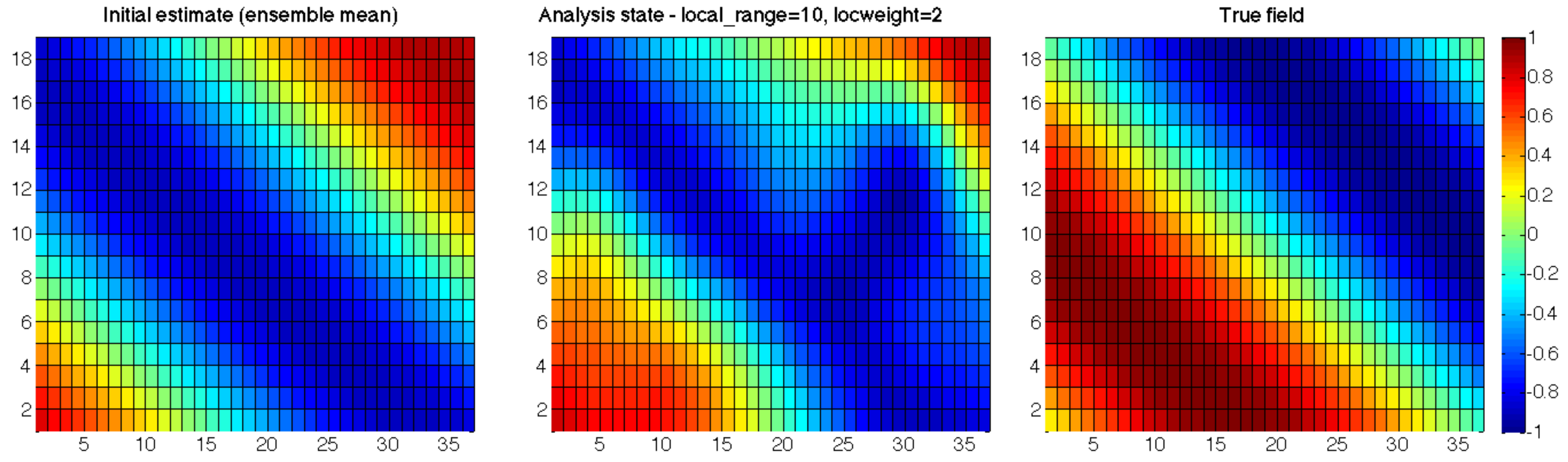


PDAF Parallel Data Assimilation Framework

Result of the local assimilation (2)

```
./PDAF_offline -filtertype 7 -cradius 10.0 -locweight 2
```

- Observation weighting by 5th-order polynomial
- Analysis field is smoother than before because of distance-weight of observations

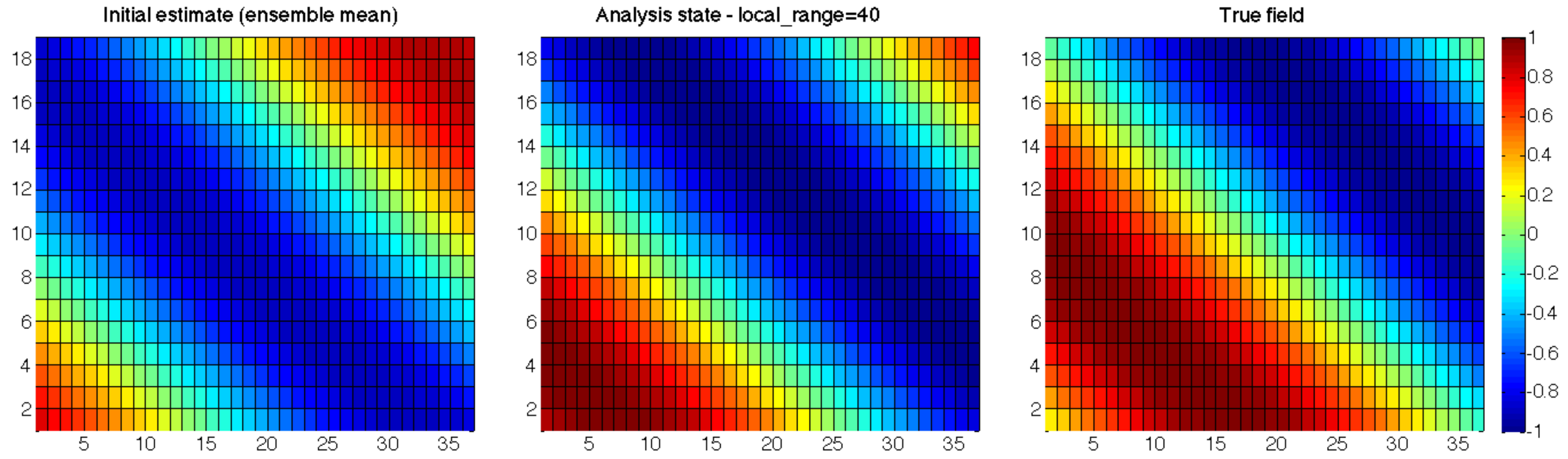


PDAF Parallel Data Assimilation Framework

Result of the local assimilation (3)

```
./PDAF_offline -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_offline.F90`

  `prepoststep_ens_offline.F90`

- Change in `init_pdaf_offline.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

$$\left.\begin{array}{l}\texttt{init\_n\_domains\_pdaf.F90}\\ \texttt{init\_dim\_l\_pdaf.F90}\end{array}\right\}$$

localize
state vector

Additional routine in callback_obs_pdafomi.F90:

$$\left.\begin{array}{l}\texttt{init\_dim\_obs\_l\_pdafomi}\end{array}\right\}$$

localize
observations

Discuss now the files in the order they are called

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_assimilation`
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_obs_l_pdafomi`

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) Declare

```
INTEGER, ALLOCATABLE :: id_lstate_in_pstate(:)
```

b) Allocate `id_lstate_in_pstate(dim_l)`

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

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

d) Provide `id_lstate_in_pstate` by calling

```
CALL PDAFlocal_set_indices(dim_l, id_lstate_in_pstate)
```

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)

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

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 cut-off radius
`locweight`	– the weighting method

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

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

PDAF Parallel Data Assimilation Framework

2 Using Parallelization

2a) Use local filter OpenMP-parallelization

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” concurrently
  - 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, operate on 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

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’

OpenMP-support is fully implemented in the templates!

PDAF Parallel Data Assimilation Framework

Running the tutorial program

Run analogously to case without parallelization

- cd to `/tutorial/offline_2D_serial`
- Set environment variable `PDAF_ARCH` or set it in Makefile (e.g. `linux_gfortran`)
- 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
```

OpenMP in the local filters

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

Settings to make OpenMP work are in: `mod_assimilation.F90`

Last line of case-specific part of `mod_assimilation.F90` is

```
!$OMP THREADPRIVATE(coords_l)
```

- This variable is specific for each local analysis domain
- The variable is declared in mod_assimilation.F90
- The declaration 'THREADPRIVATE' ensures that the variable can have a different value in the different threads

PDAF Parallel Data Assimilation Framework

2b) Parallelized global filter

Parallelize the analysis step

Implementation Strategy:

Take files from global analysis without parallelization and add the parallelization

Parallelization:

- Perform analysis step using multiple processors
- Split the state vector into equal parts to distribute the work

Notation for parallelization:

- Suffix `_p` marks variables with process-specific values

- Parallelization variables are declared in the module **`mod_parallel`**

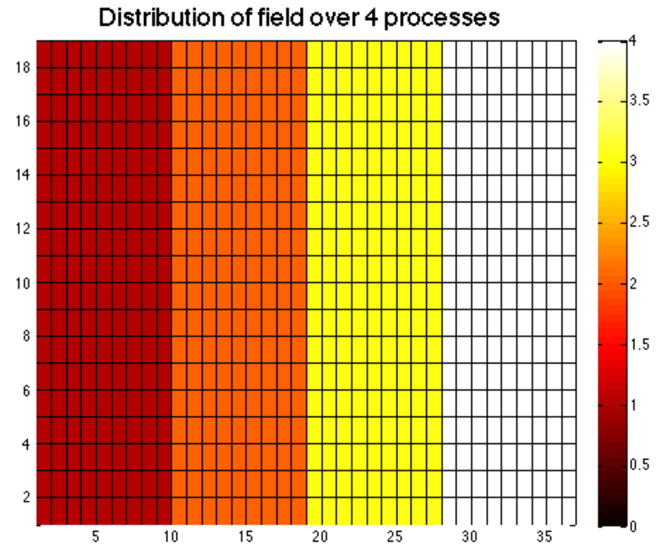
Decomposition of model field

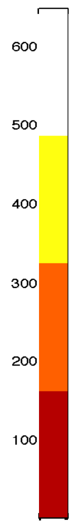
We want to distribute the state vector over the processes

→ Split state vector into approximately equal continuous parts

→ Corresponds to distribution along second index of model field (the first one in continuous in memory)

For 36 grid points we have uniform distributions for 2,3,4,6,or 9 processes (other numbers are possible)





Distribution of state vector over 4 processes

PDAF Parallel Data Assimilation Framework

Running the parallel tutorial program

- cd to /tutorial/offline_2D_parallel
- Set environment variable PDAF_ARCH or set it in Makefile (e.g. linux_gfortran)
- Clean existing files with ‘`make cleanall`’
  (This also removes the compiled PDAF library from previous tests)
- Compile by running ‘`make`’
  (this also builds the PDAF library again)
- Run the program with

```
mpirun -np X ./PDAF_offline
```

(X>0; optimal are `X=1,2,3,4,6` because then `ny=36` is dividable by X)

Impact of the parallelization

- Ensemble array is distributed ➜ less memory per process (visible in the memory display at the end of the screen output):

$ mpirun –np **1** ./PDAF_offline

```
              Allocated memory  (MB)
         state and A:    0.005 MiB (persistent)
      ensemble array:    0.044 MiB (persistent)
       analysis step:    0.027 MiB (temporary)
```

$ mpirun –np **4** ./PDAF_offline

```
             Allocated memory  (MB)
         state and A:    0.002 MiB (persistent)
      ensemble array:    0.011 MiB (persistent)
       analysis step:    0.019 MiB (temporary)
```

Note: Memory for analysis step is not changed!

Impact of the parallelization (2)

Screen output shows some influence of the parallelization

```
Parallelization - Filter on model PEs:
             Total number of PEs:     4
  Number of parallel model tasks:     1
                  PEs for Filter:     4
# PEs per ensemble task and local ensemble sizes:
          Task   1
          #PEs   4
             N   9
```

At analysis step:

```
 --- PE-domain   1 dimension of observation vector  8
 --- PE-domain   2 dimension of observation vector  8
 --- PE-domain   3 dimension of observation vector  8
 --- PE-domain   4 dimension of observation vector  4
```

Note: The output lines might be unordered

PDAF Parallel Data Assimilation Framework

Global ESTKF: Files to be changed for parallelization

```
mod_assimilation.F90
initialize.F90
~~init_pdaf_offline.F90~~
init_ens_offline.F90
obs_A_pdafomi.F90
~~callback_obs_pdafomi.F90~~
prepoststep_ens_offline.F90
```

No change: ~~init_pdaf_offline.F90~~

No change: ~~callback_obs_pdafomi.F90~~

mod_assimilation.F90 – Fortran module

initialize.F90, init_pdaf_offline.F90, init_ens_offline.F90 – initialization

obs_A_pdafomi.F90, callback_obs_pdafomi.F90 – analysis step

prepoststep_ens_offline.F90 – post step

PDAF Parallel Data Assimilation Framework

initialize.F90 – parallelization

Initialize the model information – we have: `nx, ny, dim_state_p`

1. Use additional dimensions from `mod_assimilation`:

```
integer :: dim_state
integer, allocatable :: local_dims(:)
```

2. Rename `dim_state_p` to `dim_state` (global dimension)
3. Allocate `local_dims(npes_model)`
4. Set `dim_state_p` and `local_dims(:)`
   – distribute `dim_state` over number of processes

```
local_dims = FLOOR(REAL(dim_state) / REAL(npes_model))
DO i = 1, (dim_state - npes_model * local_dims(1))
   local_dims(i) = local_dims(i) + 1
END DO

dim_state_p = local_dims(mype_model+1)
```

init_ens_offline.F90 – parallelization

Initialize ensemble matrix `ens_p`

Simple parallel variant:

1. Initialize global ensemble array (only one process)
2. Distribute sub-states of ensemble array
   (from the process doing step 1 to all others)

1. Required steps – only for `mype_filter==0`

- Declare array `ens` and
  allocate `ens(dim_state, dim_ens)`
- Use serial implementation for initialize **`ens`**
  (replace `ens_p` by `ens`)

init_ens_offline.F90 – parallelization (2)

2. Distribute sub-states of ensemble array

For `mype_filter=0`

a) Initialize local part of `ens_p` directly:

```
ens_p(1:dim_p,1:dim_ens) = ens(1:dim_p,1:dim_ens)
```

b) Distribute other sub ensembles

```
DO domain=2, npes_filter
     allocate ens_p_tmp(local_dims(domain), dim_ens)
     fill ens_p_tmp with part of ens for domain
     MPI_Send ens_p_tmp from process 0 to process 'domain-1'
     deallocate ens_p_tmp
```

PDAF Parallel Data Assimilation Framework

init_ens_offline.F90 – parallelization (3)

2. Distribute sub-states of ensemble array

For all processes with `mype_filter>0`:

MPI_Recv `ens_p_tmp` into **`ens_p`**

Notes:

- "Classical" MPI communication: MPI_Send/MPI_Recv
- See tutorial code for MPI function calls
- Offset in state vector for mype_filter=k is

  sum of `local_dims(i)` from i=1 to k
- Size of state vector part is `local_dims(k)`
- The example code is not the most efficient possibility:
  Each process could read its own local part of `ens_p`

PDAF Parallel **Data Assimilation** Framework

init_dim_obs_A in obs_A_pdafomi.F90

Operations in case of parallelization

- read observation file
- count number of available observations for process-local part of state vector (direct output to PDAF: **`dim_obs_p`**)
- initialize array holding process-local available observations
- initialize array of index of observation in process-local state vector
- Call `PDAFomi_gather_obs` to finalize initializations

Adapt serial implementation for these operations

PDAF Parallel Data Assimilation Framework

init_dim_obs_A - parallelization (1)

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

1. Get offset of local part in global state vector

   `off_p` = Sum over local_dims(i) up to i=mype_filter

2. Now count

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

PDAF Parallel Data Assimilation Framework

init_dim_obs_A - parallelization (2)

Initilialize `obs_p` and `obs_index_p` (now process-local parts)

```
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+local_dims(mype_filter+1)) 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
        ocoord_p(1, cnt_p) = REAL(j)      ! X-coordinates
        ocoord_p(2, cnt_p) = REAL(i)      ! Y-coordinates
    END IF; END IF
  END DO
END DO
```

preposstep_ens_offline.F90 – parallelization

Post-step routine for the offline mode

Adapt writing of output files for parallelism

ensemble array `ens_p` is distributed

To do – inverse operations to `init_ens_offline`

- Use temporary array `ens_p_tmp`
- For mype_filter>0:
  - *MPI_Send* `ens_p` to mype_filter=0
- For mype_filter=0:
  - Do domain=2, npes_filter
  - *MPI_Recv* into `ens_p_tmp`
  - Initialize part of global array `ens` with `ens_p_tmp`
  - Write `ens` into files

prepoststep_ens_offline.F90 – parallelization (2)

Also in the tutorial implementation

- Collect local mean states (`state_p`) into a global analysis state and write to file.
- Collect vector of estimated variance (`variance_p`) into a global variance vector. Compute estimated RMS error from it.

Done!

The analysis step in offline mode with parallelization is fully implemented now

The implementation allows you now to use the global filters ESTKF, ETKF, EnKF, and SEIK

2c) Parallelized local filter

Impact of the parallelization

- Ensemble array is distributed ➜ less memory per process (visible in the memory display at the end of the screen output):

```
$ mpirun –np 1 ./PDAF_offline –filtertype 7
```

```
            Allocated memory  (MB)
       state and A:    0.010 MiB (persistent)
    ensemble array:    0.044 MiB (persistent)
     analysis step:    0.020 MiB (temporary)
```

```
$ mpirun –np 4 ./PDAF_offline –filtertype 7
```

```
            Allocated memory  (MB)
       state and A:    0.003 MiB (persistent)
    ensemble array:    0.011 MiB (persistent)
     analysis step:    0.020 MiB (temporary)
```

Note: Memory for analysis step is not changed!

PDAF Parallel Data Assimilation Framework

Impact of the parallelization (2)

Screen output shows some influence of the parallelization

```
 Parallelization - Filter on model PEs:
                 Total number of PEs:     4
 …
```

At analysis step:

```
PDAF      --- local analysis domains(min/max/avg):    162   162 162.0
…
PDAFomi          --- Number of full observations       28
```

Parallelize the local analysis step

Take files from

- global analysis with parallelization and
- localized analysis without parallelization

and adapt

Parallelization:

- Perform analysis step using multiple processors
- Split the state vector into equal parts to distribute the work
    - As we did for the global filter

Notation for parallelization:

- Suffix `_p` marks variables with process-specific values

Local filter LESTKF – parallelization

Required files to be parallelized

`init_n_domains_pdaf.F90` } Needs adaption

~~`callback_obs_pdafomi.F90`~~ } From local filter – no changes

~~`obs_A_pdafomi.F90`~~

~~`init_dim_obs_A`~~

~~`obs_op_A`~~

} From **parallel global filter** – no changes

~~`init_dim_obs_l_A`~~ } From local filter – no changes

`init_dim_l_pdaf.F90` } Needs adaption in coordinates

init_n_domains_pdaf.F90

Routine to set the number of local analysis domains

`n_domains_p`: now the number of local analysis domains for the particular process (according to part of state vector)

To do:

1. Include `local_dims` with `use mod_assimilation`
2. Set

```
n_domains_p = local_dims(mype_filter+1)
```

init_dim_l_pdaf.F90

Routine to set the local state dimension, local coordinates and indices

`coords_l`: Still the coordinates of the local analysis domain in the full model domain

To do:

1. Determine offset of `domain_p` due to parallelization

```
off_p = 0
DO i = 1, mype_filter
      off_p = off_p + local_dims(i)
END DO
```

2. Compute coordinates accounting for offset

coords_l(1) = REAL(CEILING(REAL(domain_p+off_p)/REAL(ny)))

coords_l(2) = REAL(domain_p+off_p) - (coords_l(1)-1)*REAL(ny)

Done!

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

The implementation allows you now to use all local filters!
(LESTKF, LETKF, LSEIK, LNETF, LKNETF)

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)

3) Hints for adaptions for real models

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 routines handling observations
- Adapt file output:
    - need to read and write restart files from specific model
    - adapt writing of ensemble mean state in `prepoststep_pdaf`

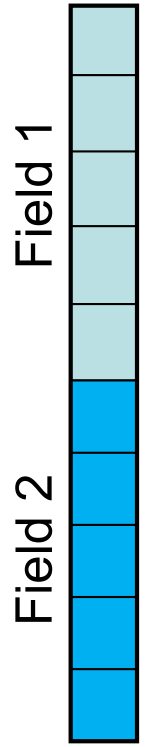
Multiple fields in state vector

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

  → see tutorial for online mode with serial model for example of 2 fields (`online_2D_serialmodel_2fields`)

- For two or more fields:
  - concatenate them in the state vector
  - 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 thisobs%id_obs_p) for correct offset of observed field in state vector

State vector with 2 fields



PDAF Parallel Data Assimilation Framework

Multiple observed fields

- In tutorial:
  - We discussed observations of one field at some grid points
  - 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`)

The End!

Tutorial described example implementations

- Offline mode of PDAF
- Simple 2D example
- Implementation supports various filters
  - global and with localization
  - without and with parallelization
- 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