The Parallel Data Assimilation Framework PDAF for scalable sequential data assimilation

Lars Nerger

Alfred Wegener Institute for Polar and Marine Research
Bremerhaven, Germany

and

Bremen Supercomputing Competence Center BremHLR

lars.nerger@awi.de

BremHLR
Kompetenzzentrum für Höchstleistungsrechnen Bremen

Overview

Focus on computational aspects of data assimilation

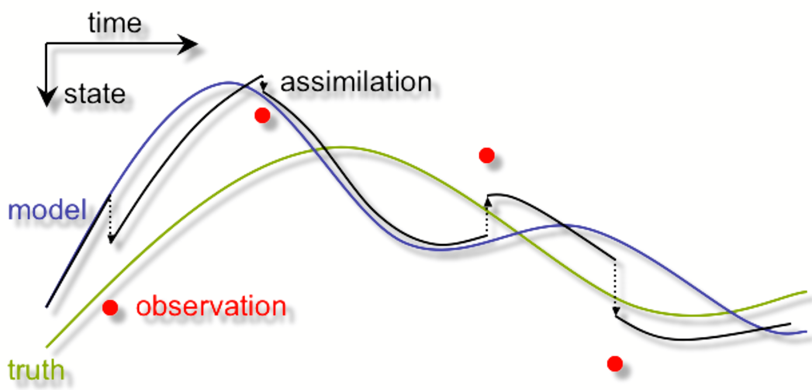
- Sequential data assimilation
- Parallel Data Assimilation Framework PDAF
- Parallel performance with PDAF

Sequential Data Assimilation

Sequential Data Assimilation

Goal

Combine model and observations for improved state representation



Method

Iteration:

Analysis:
Correct model state estimate when observations are available.

⟷

Forecast:
Propagate state and error estimate

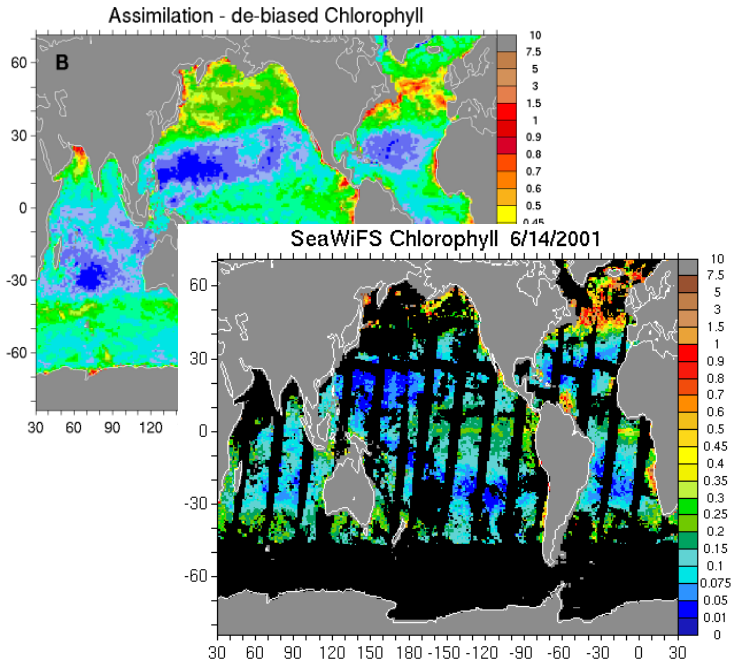
Common sequential algorithms

- Ensemble-based Kalman filters
- Particle filters

Application examples

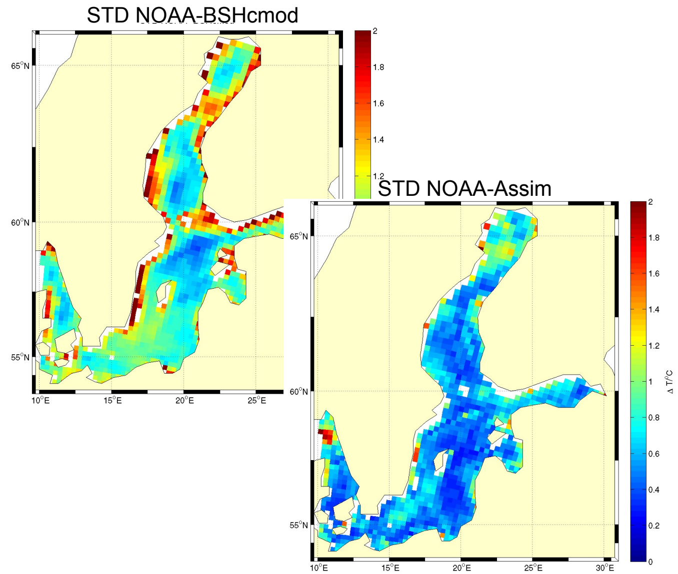
Ocean chlorophyll assimilation into NASA Ocean Biogeochemical Model (with Watson Gregg, NASA GSFC)

- Generation of daily re-analysis maps of chlorophyll at ocean surface
- Work toward multivariate assimilation



Coastal assimilation of ocean surface temperature (project “DeMarine Environment”, AWI and BSH)

- North Sea and Baltic Sea
- Improve operational forecast skill, e.g. for storm surges



Computational and Practical Issues

Memory: Huge amount of memory required
(model fields and ensemble matrix)

Computing: Huge requirement of computing time
(ensemble integrations)

Parallelism: Natural parallelism of ensemble integration exists
- but needs to be implemented

Implementation: Existing models often not prepared for data assimilation

„Fixes“: Filter algorithms need „fixes“ and tuning
(literature provides typical methods)

Parallel Data Assimilation Framework

Models and Filter Algorithms

- Sequential assimilation algorithms require limited information
  - no physics needed!
  - relation of model fields to state vector
  - observations (time, type, location, error)

Because of this:

- Filter algorithms can be developed and implemented independently from model
- Model can be developed independently from the filter
- Parallelization of ensemble forecast can be implemented independently from model

Motivation for a Framework

A framework allows to

- Provide fully implemented parallelized and optimized filter algorithms
- Provide collection of „fixes“, which showed good performance in studies
- Provide parallelization support (parallel environment) for ensemble forecasts
- Provide uniform interface for a model to data assimilation

- Simplify implementation of data assimilation systems with existing models

Online and Offline modes

Offline

- Separate executable programs for model and filter
- Ensemble forecast by running sequence of models
- Analysis by filter program
- Data exchange model-filter by files on disk

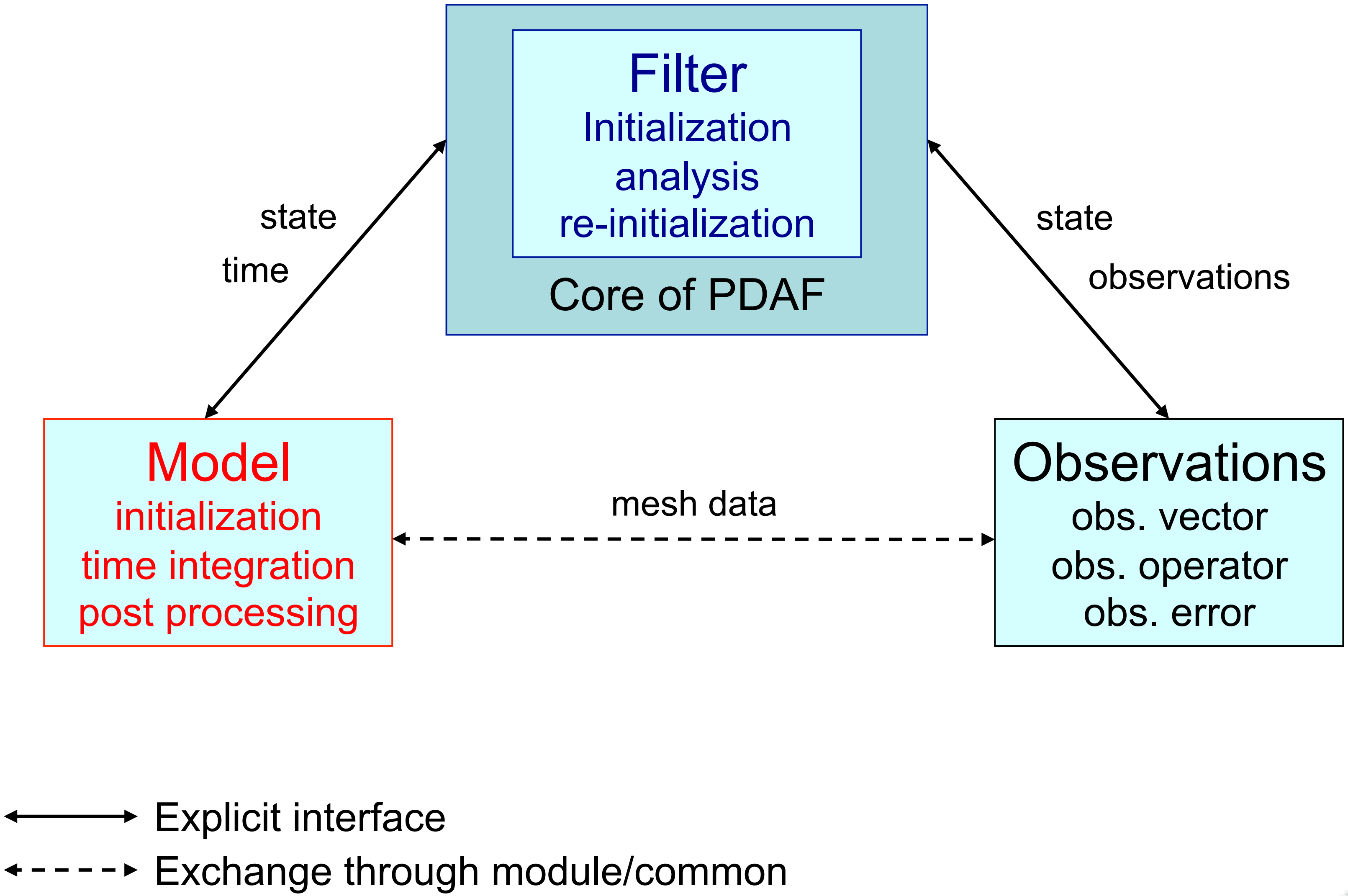
Online

- Couple model and filter into single executable program
- Run one program for whole assimilation task (forecasts and analysis)

- *Advantage:* Rather easy implementation (file reading/ writing routines, no change to model code)
- *Disadvantage:* Limited efficiency

- *Disadvantage:* More implementation work, incl. extension of model code.
- *Advantage:* Computationally very efficient

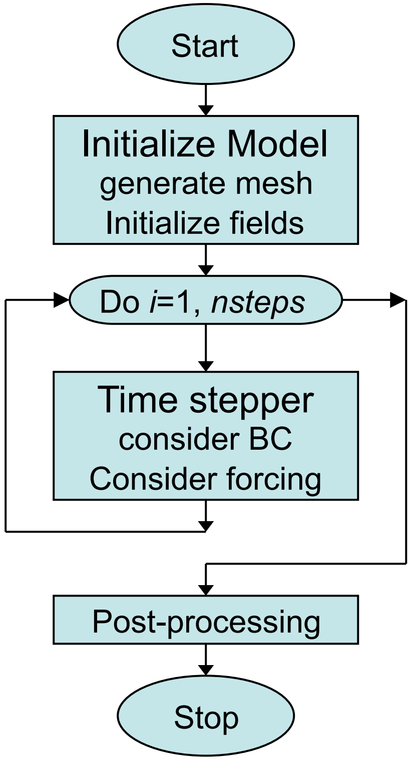
PDAF: Logical separation of assimilation system

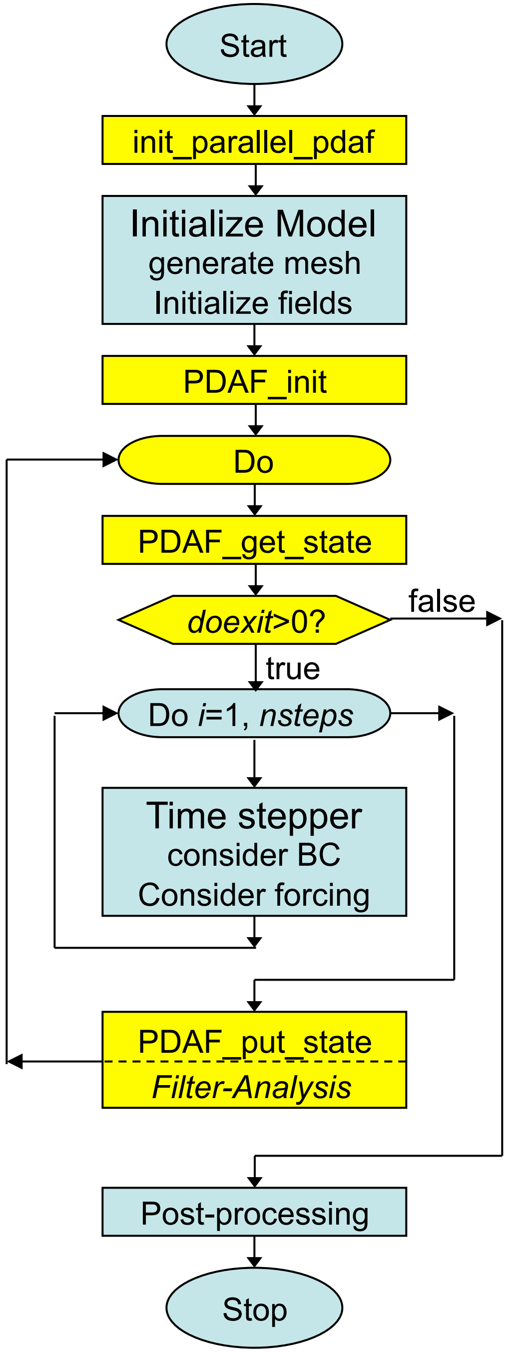


PDAF: Design considerations

- Combination of filter with model with minimal changes to model code
- No subroutine-requirement for model
- Control of assimilation program coming from model
- Easy switching between different filters
- Easy switching between different observational data sets
- Complete parallelism in model, filter, and framework

Model





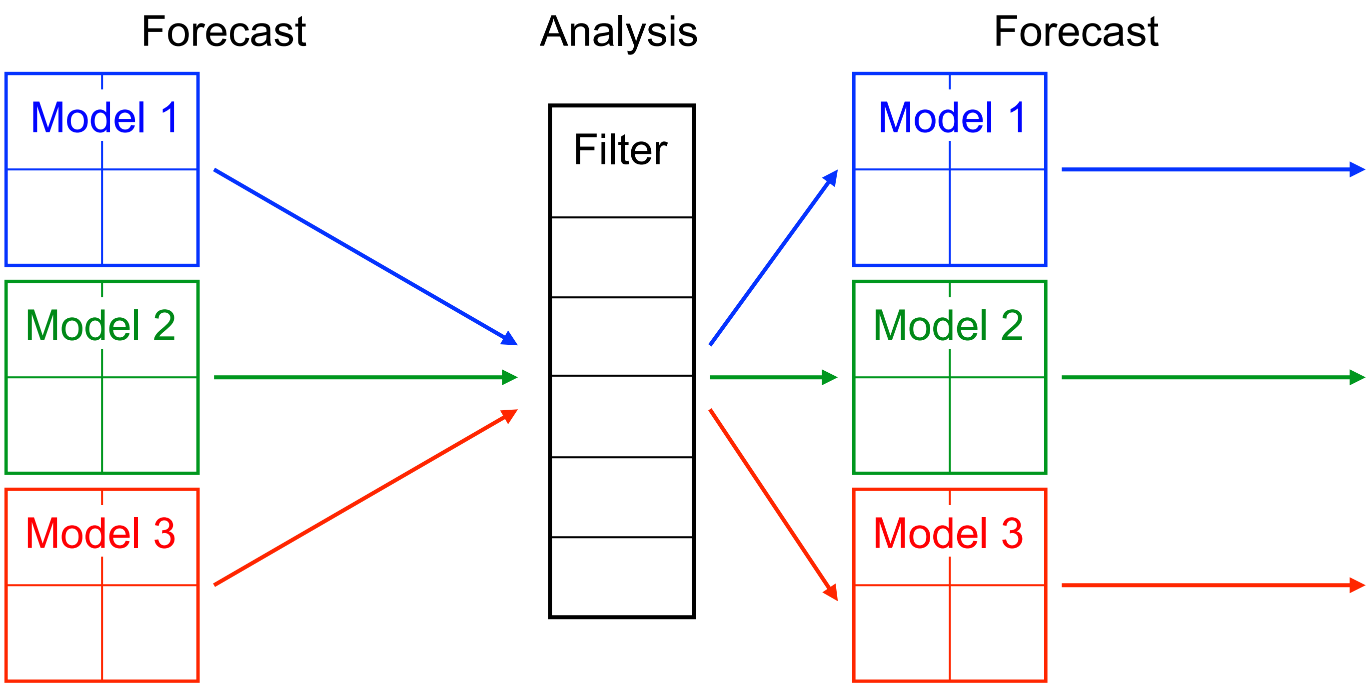
Extension for data assimilation

External Do-loop an be avoided – less flexibility!

PDAF Standard Interface

- Interface independent of filter
  (except for names of user-supplied subroutines)
- Plain calls to subroutines with basic data types
- User-supplied routines for elementary operations:
  - field transformations between model and filter
  - observation-related operations
  - filter pre/post-step
- User supplied routines can be implemented
  as routines of the model
  (e.g. share common blocks or modules)
- Model-sided configuration of assimilation system
- Low abstraction level for optimal performance

2-level Parallelism



1. Multiple concurrent model tasks
2. Each model task can be parallelized

➢ Analysis step is also parallelized

Existing Online Implementations

- FEOM (Finite-Element Ocean Model)
  - PDAF’s “home” model; all features
- MIPOM (met.no, by I. Burud)
  - First implementation not done by myself
- NOBM (NASA Ocean-Biogeochemical Model)
  - For ocean-color assimilation
- BSHcmod (Project DeMarine Environment)
  - Toward operational use in North/Baltic Seas
- ADCIRC (at KAUST, I. Hoteit, with Umer Altaf)
  - 3 days for basic implementation

Filter algorithms in PDAF

Implementations mostly from filter-comparison studies

- Ensemble Kalman filter (EnKF, Evensen, 1994)
- SEEK filter (Pham et al., 1998a)
- SEIK filter (Pham et al., 1998b)
- ETKF (Bishop et al., 2001)
- LSEIK filter (Nerger et al., 2006)
- LETKF (Hunt et al., 2007)
- EnSKF (Whitaker & Hamill, 2002)
- LSEIK with OBC (Nerger/Gregg, 2008)

with localization

Software aspects

- Language: Fortran95
  - Motivated by ocean circulation models
  - Can be compiled and linked as a library
- Parallelization: MPI
- Required Libraries: BLAS & LAPACK
- For compilation: make

- Compilation and execution verified on many different machines (from notebook to supercomputer)

PDAF is available!

- Open source
- Web site

pdaf.awi.de

- Code download
- Documentation wiki
- Distributed is the source code of PDAF together with an example implementation

Parallel Performance of PDAF

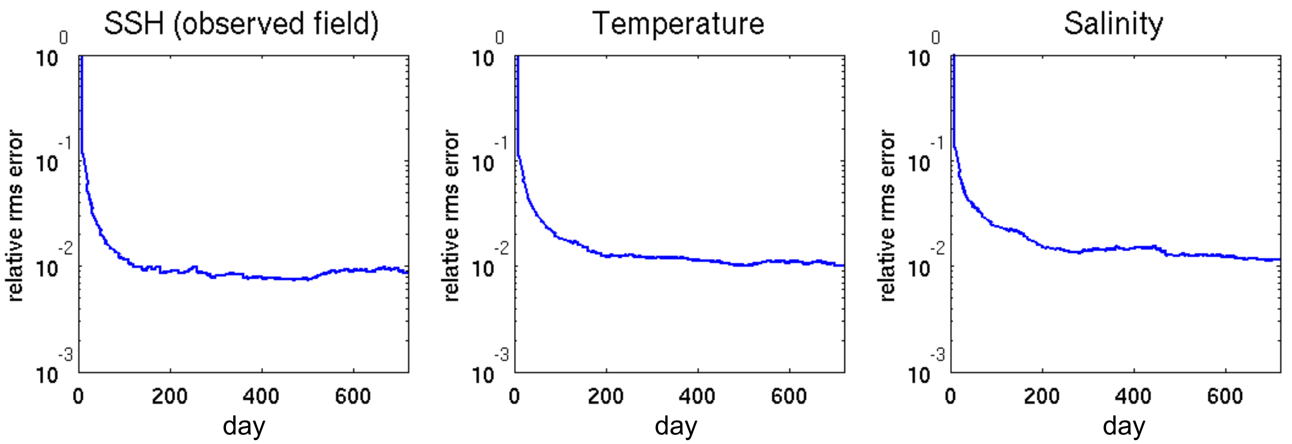
Application Example

Test case: „Twin Experiment“

- FEOM (Finite Element Ocean Model)
- North Atlantic, 1 degree resolution, 20 z-levels (small mesh)
- Assimilate synthetic sea level observations over 2 years
- Data available each 10 days; all grid points

Assimilation impact

improve model fields by 2 orders of magnitude



Parallel performance of PDAF

- Performance tests on

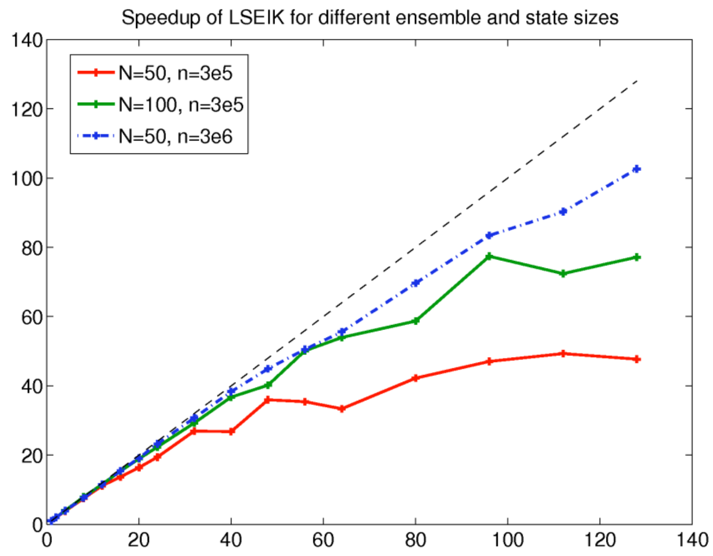
  SGI Altix ICE at HRLN (German “High performance computer north”)

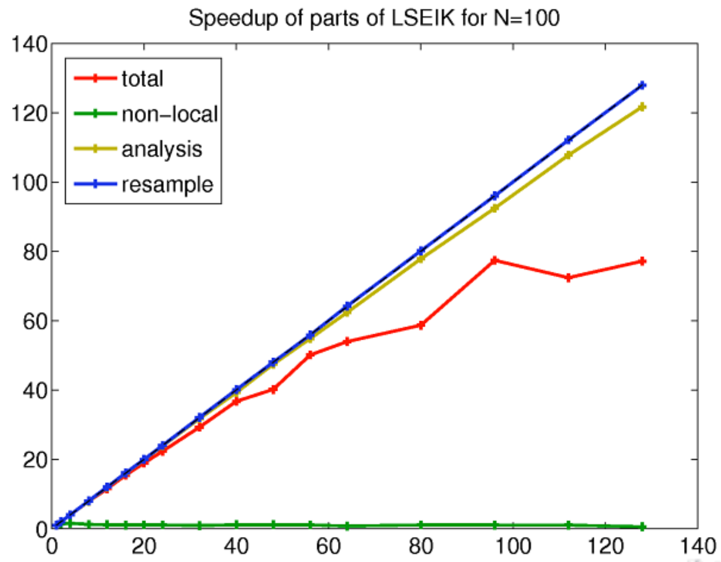
  nodes: 2 quad-core Intel Xeon Gainestown at 2.93GHz
  network: 4x DDR Infiniband
  compiler: Intel 10.1, MPI: MVAPICH2

- Ensemble forecasts
  - are naturally parallel
  - dominate computing time
    Example: parallel forecast over 10 days: 45s
    SEIK with 16 ensemble members: 0.1s
    LSEIK with 16 ensemble members: 0.7s

Speedup of LSEIK with domain decomposition

- LSEIK performs sequence of local optimizations on sub-subdomains defined by influence radius for observations
  - near-ideal speedup for analysis step and resampling (ensemble transformation)
  - total speedup is limited by
    - non-local gathering of observation-state residuals
    - pre/poststep





State dimension n = 300,000
Observations m = 30,000
Ensemble size N

Parallel Performance

Use between 64 and 4096 processors of SGI Altix ICE cluster (Intel processors)

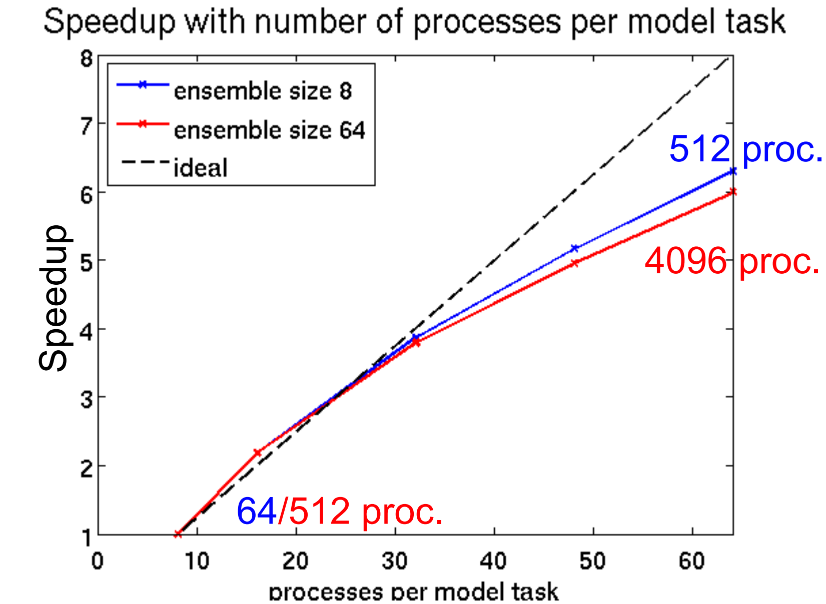
94-99% of computing time in model integrations

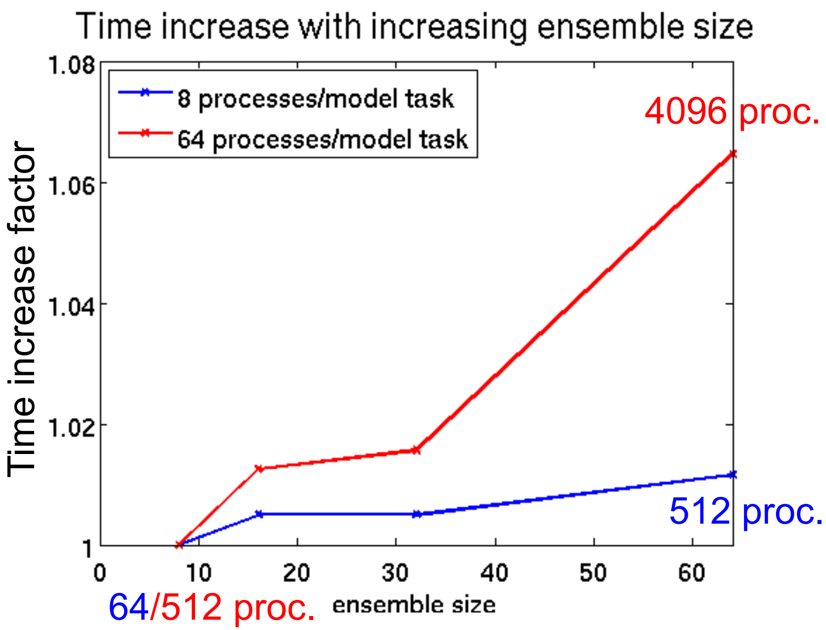
Speedup: Increase number of processes for each model task, fixed ensemble size

- factor 6 for 8x processes/model task
- one reason: time stepping solver needs more iterations

Scalability: Increase ensemble size, fixed number of processes per model task

- increase by ~7% from 512 to 4096 processes (8x ensemble size)
- one reason: more communication on the network





Summary

PDAF provides

- Simplified implementation of assimilation systems
- Flexibility: Different assimilation algorithms and data configurations within one executable
- Full utilization of parallelism in models and filters
- Good scalability for large-scale systems

http://pdaf.awi.de

Thank you!