Advancing the TOPAZ ocean and sea ice data assimilation system: Algorithmic innovations enabled by NEDAS

Yue (Michael) Ying NERSC

Supported by: SASIP SCHMIDT FUTURES ACCIBERG European Commission

EnKF Workshop 2025, Ullensvang, Norway

Data assimilation software at NERSC

Previously, each model has **a specific DA software** separately maintained:

NorCPM: NorESM  with DEnKF, EnOI, …

TOPAZ: HYCOM-CICE  with DEnKF

BIORAN: HYCOM-ECOSMO  with EnKS

neXtSIM-f: neXtSIM  with nudging

NEDAS now provides support to **all NERSC models**

- **Rapid research development**
- **Access to rich Python ecosystem** for scientific computation, machine learning etc.

NorESM, HYCOM-CICE, neXtSIM, ECOSMO, WRF, …

NEDAS

EnKF (different flavors: ETKF or EAKF), EnKS, non-Gaussian filters, alignment, emulators…

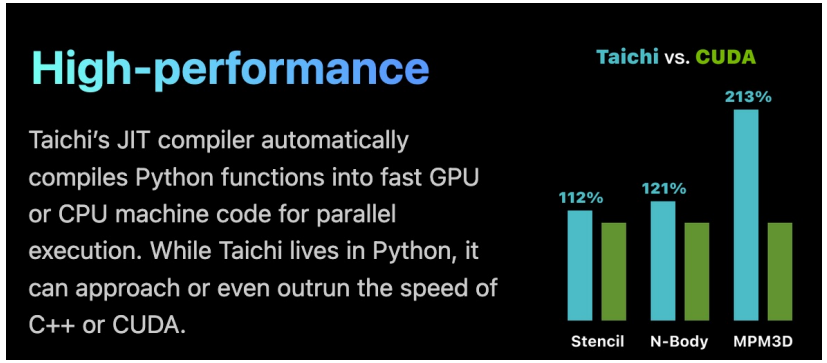
NumPy Numba OpenCV TensorFlow K Keras

Python tools for high-performance computation

- NumPy drives most of the linear algebra
- mpi4py provides support for MPI parallelism
- JIT technique (numba, JAX, taichi) to convert python functions to machine code

Veros

Versatile Ocean Simulation in Pure Python




```python
@numba.njit
def obs_increment_eakf(obs_prior, obs, obs_err) -> np.ndarray:
    nens = obs_prior.size

    ##obs error variance
    obs_var = obs_err**2

    ##obs_prior separate into mean+perturbation
    obs_prior_mean = np.mean(obs_prior)
    obs_prior_pert = obs_prior - obs_prior_mean

    ##compute prior error variance
    obs_prior_var = np.sum(obs_prior_pert**2) / (nens-1)

    """ensemble adjustment Kalman filter (Anderson 2003)"""
    var_ratio = obs_var / (obs_prior_var + obs_var)

    ##new mean is weighted average between obs_prior_mean and obs
    obs_post_mean = var_ratio * obs_prior_mean + (1 - var_ratio) * obs

    ##new pert is adjusted by sqrt(var_ratio), a deterministic square-r
    obs_post_pert = np.sqrt(var_ratio) * obs_prior_pert

    ##assemble the increments
    obs_incr = obs_post_mean + obs_post_pert - obs_prior

    return obs_incr
```

NERSC

Python as control scripts (thanks to “Code Clinic” by ICCS)

- YAML configuration files

```yaml
nens: 30
run_analysis: True
run_diagnose: True
debug: True

time_start: 2021-07-05T00:00:00Z
time_end: 2021-08-09T00:00:00Z
time_analysis_start: 2021-07-12T00:00:00Z
time_analysis_end: 2021-08-09T00:00:00Z
cycle_period: 168

model_def:
  topaz.v5:
    config_file: '{nedas_root}/models/topaz/v5/default.yml'
    model_env: '{nedas_root}/models/topaz/env/setup.src'
    reanalysis_code: '/cluster/home/yingyue/code/ReanalysisTP5'
    basedir: '/cluster/work/users/yingyue/TP5a0.06'
    nproc_per_run: 512
    nproc_per_util: 50
    walltime: 3600
    restart_dt: 168
    forcing_dt: 6
    ens_run_type: scheduler
    use_job_array: False
    stagnant_log_timeout: 100
    ens_init_dir: '/cluster/work/users/yingyue/26118/FORECAST'
    forcing_file: '/cluster/work/users/yingyue/TP5a0.06/force/syr
    truth_dir: ''

analysis_scheme: offline_filter

assimilator_def:
  type: TopazDEnKF
```

- Better error handling
- Multiplatform support: HPC, jupyter-lab, containers

```python
##build the shell command line
model_exe = os.path.join(self.basedir, f'expt_{self.X}', 'build', f'src_{self.V}
shell_cmd = ""
if self.model_env:
    shell_cmd =  ". "+self.model_env+"; "  ##enter topaz5 env
shell_cmd += "cd "+run_dir+"; "              ##enter run directory
shell_cmd += 'JOB_EXECUTE '+model_exe+" >& run.log"

##run the model, give it 3 attempts
for i in range(3):
    try:
        run_job(shell_cmd, job_name='topaz5', run_dir=run_dir,
                nproc=self.nproc, offset=task_id*self.nproc_per_run,
                walltime=self.walltime, log_file=log_file, **kwargs)
    except RuntimeError as e:
        print(f"{e}, retrying ({2-i} attempts remain)")
        run_command(f"cp {log_file} {log_file}.attempt{i}")
        continue
    ##check output
    if find_keyword_in_file(log_file, 'Exiting hycom_cice'):
        run_success = True
        break
```

NERSC

The offline filter (using model restart files)

```
class OfflineFilterAnalysisScheme:

    def filter(self, c):

        for c.iter in range(c.niter):
            c.transform_funcs = assim_tools.transforms.get_transform_funcs(c)
            state = assim_tools.state.get_state(c)
            timer(c)(state.prepare_state)(c)

            obs = assim_tools.obs.get_obs(c, state)
            timer(c)(obs.prepare_obs)(c, state)
            timer(c)(obs.prepare_obs_from_state)(c, state, 'prior')

            assimilator = assim_tools.assimilators.get_assimilator(c)
            timer(c)(assimilator.assimilate)(c, state, obs)

            updator = assim_tools.updators.get_updator(c)
            timer(c)(updator.update)(c, state)
```

for s in 1, …, N_s:

$$\mathbf{x}_s^b = F_s(\mathbf{x}^b)$$

$$\mathbf{y}_s^o = F_s^o(\mathbf{y}^o)$$

$$\mathbf{y}_s^b = F_s^o\left[\mathcal{H}\left(\mathbf{x}^b\right)\right]$$

$$\delta\mathbf{x}_s = A\left(\mathbf{x}_s^b, \mathbf{y}_s^b, \mathbf{y}_s^o, \mathbf{R}_s, \boldsymbol{\rho}_s\right)$$

$$\mathbf{x}^b = U\left(\mathbf{x}^b, \delta\mathbf{x}_s\right)$$

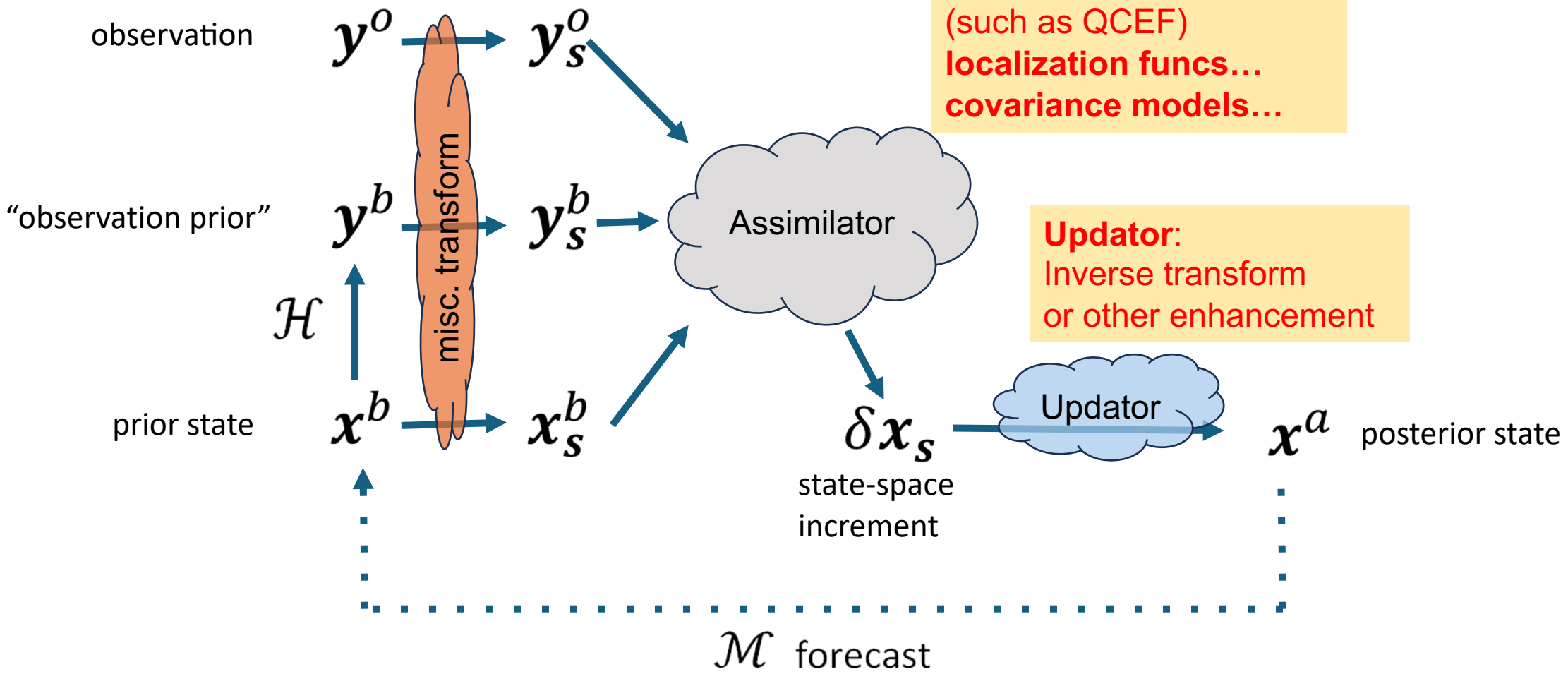
$$\mathbf{x}^a \leftarrow \mathbf{x}^b$$

NERSC

Analysis schemes

Misc. transform of prior state/obs
such as scale decomposition, anamorphosism…

Assimilator:
EnKF (different variants)
other non-Gaussian filters
(such as QCEF)
localization funcs…
covariance models…



Updator:
Inverse transform
or other enhancement

NERSC

Assimilator Classes

SearialAssimilator

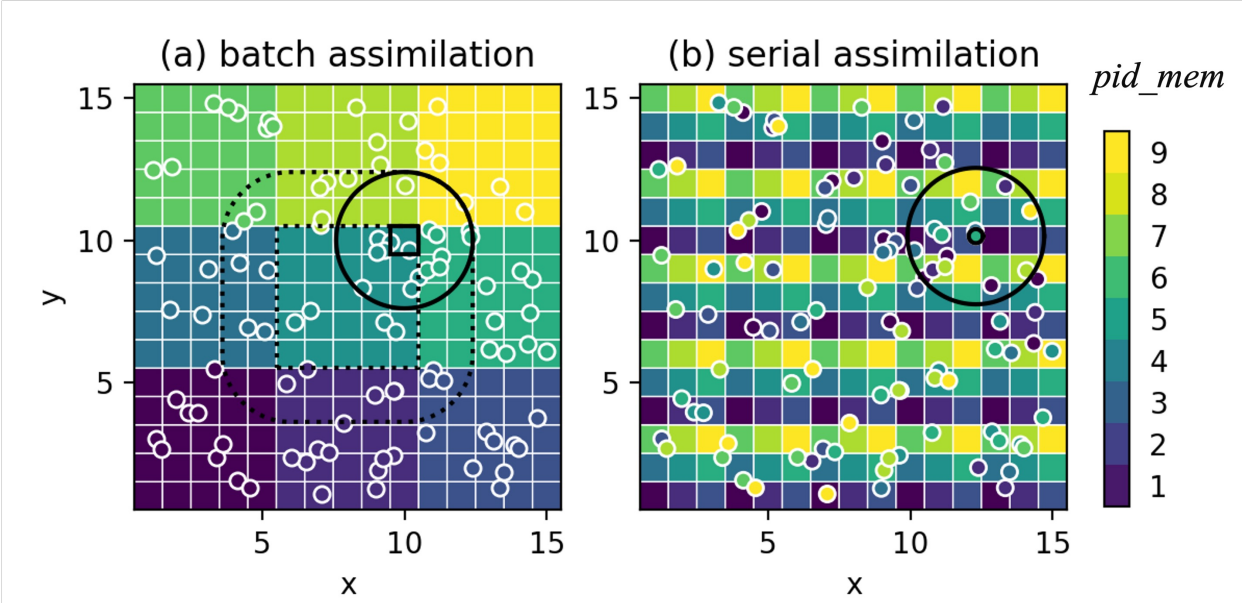
BatchAssimilator

EAKFAssimilator

QCEFAssimilator

ETKFAssimilator

TopazDEnKFAssimilator
(Exact replica of TOPAZ)



(Ying, JAMES, in review)

NERSC

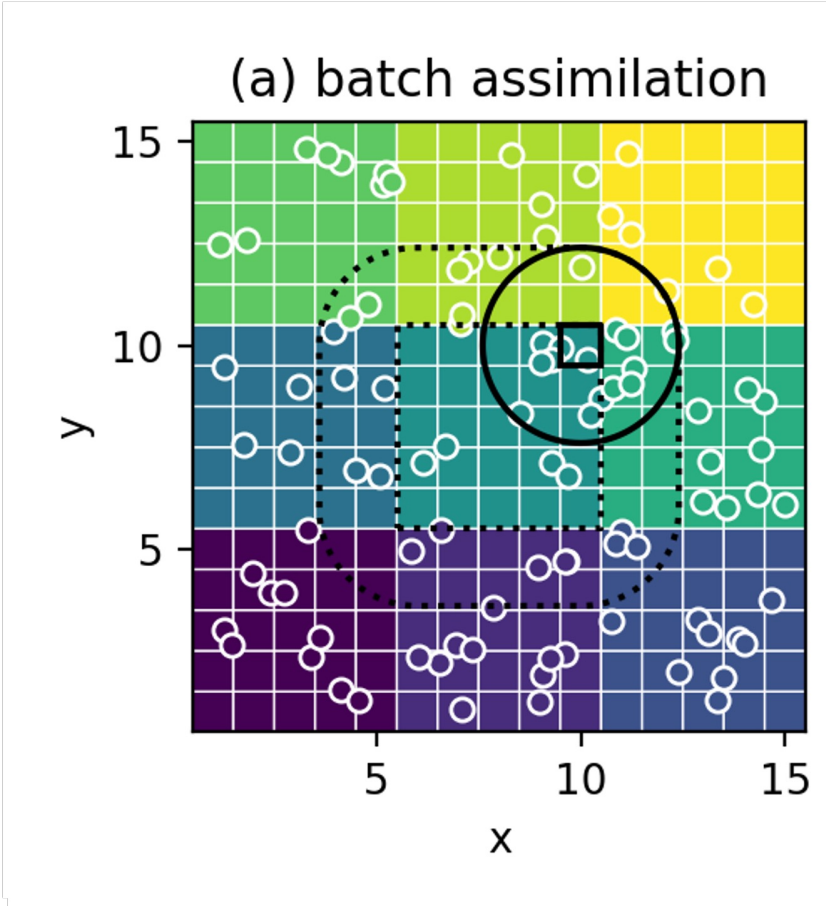
TopazDEnKFAssimilator

DEnKF (Sakov et al. 2012):
100-member ensemble without vertical localization:
Compute ensemble transform weights once, applies it to 50 levels

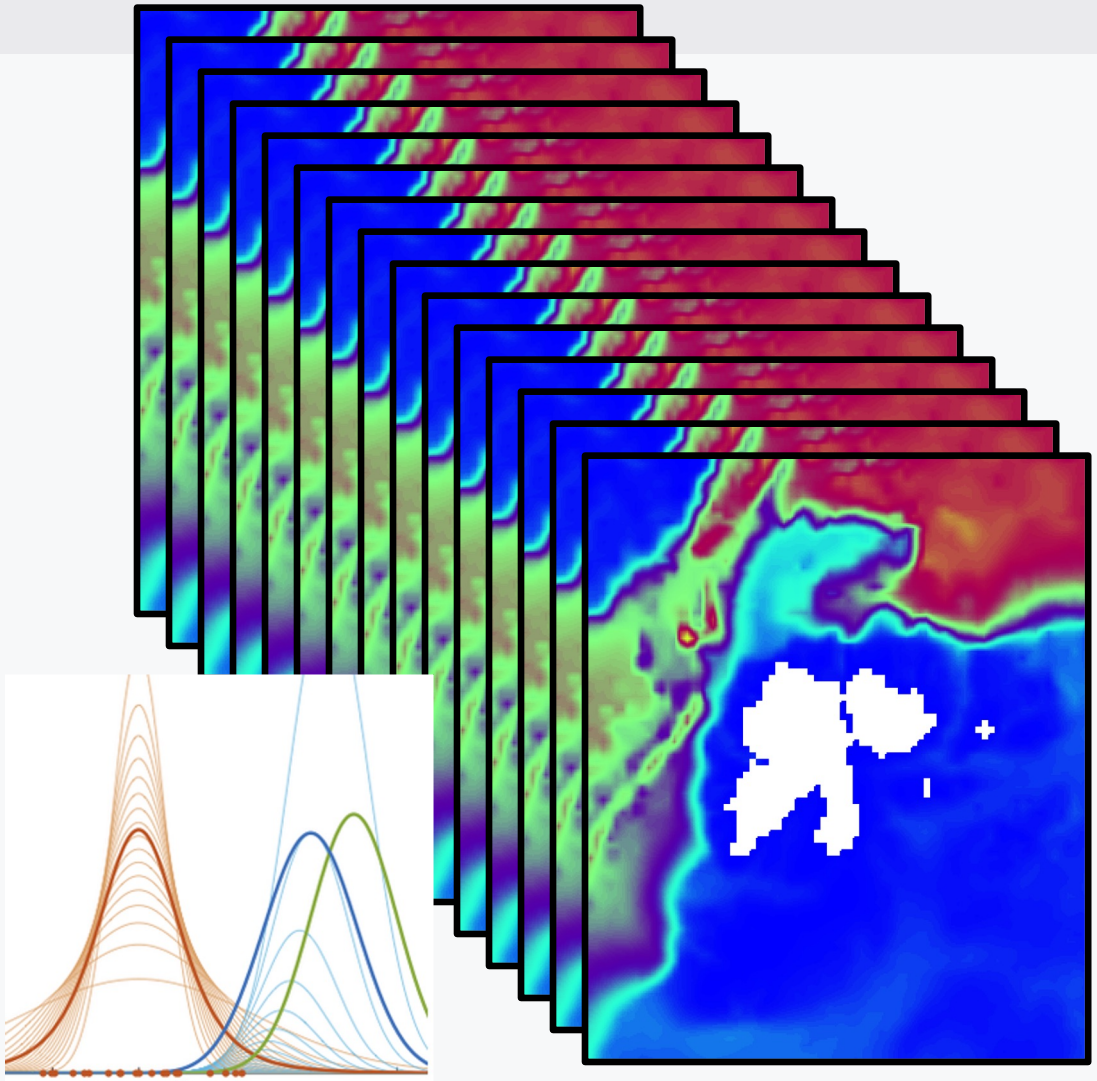
In NEDAS (ETKF available):
lfactor = hlfactor * vlfactor * tlfactor * impact_on_state

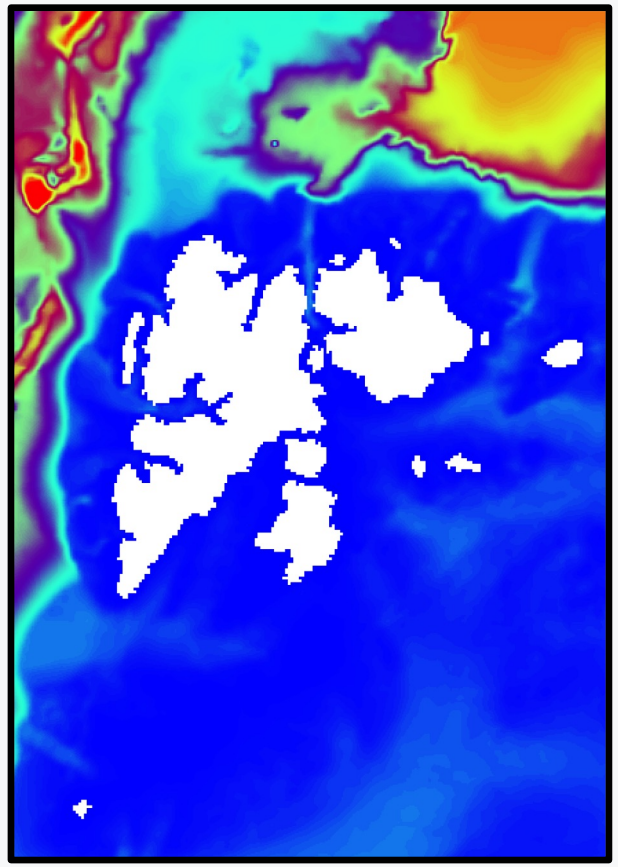
Exact replica of TOPAZ with similar runtime efficiency:
- Cache the transform weights for same localization factor
- Early exit when localization factor is zero

Computing the distance itself can be costly:
- Use L1 distance for subsetting, then refine with L2 distance
- How to keep distance-free types of algorithm efficient?



Uncertainty estimates versus model resolution – the dilemma





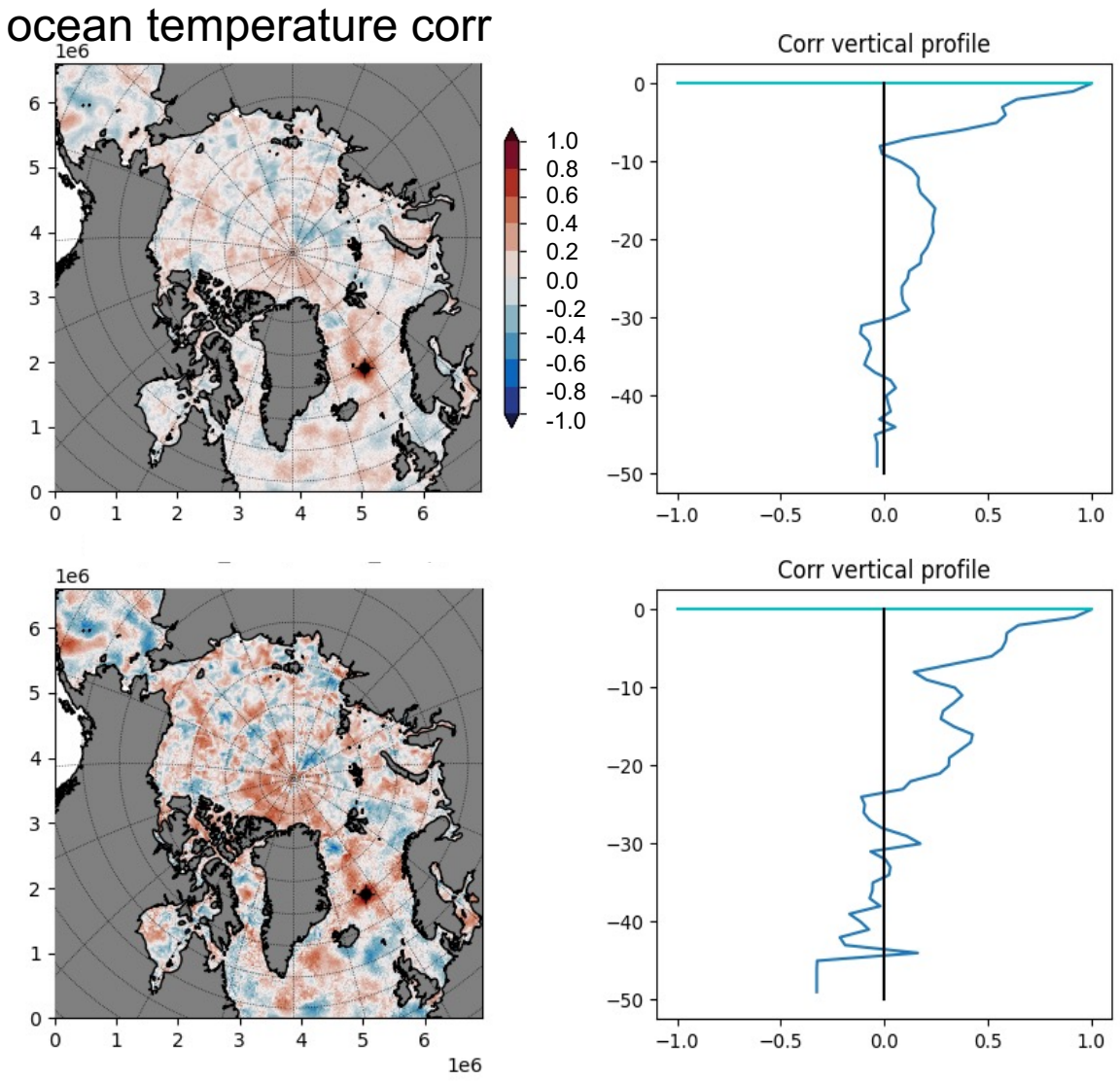
30-100 ensemble members = factor 3-4 in resolution

image credit:
Laurent Bertino

Do we need vertical localization in TOPAZ?

100 member

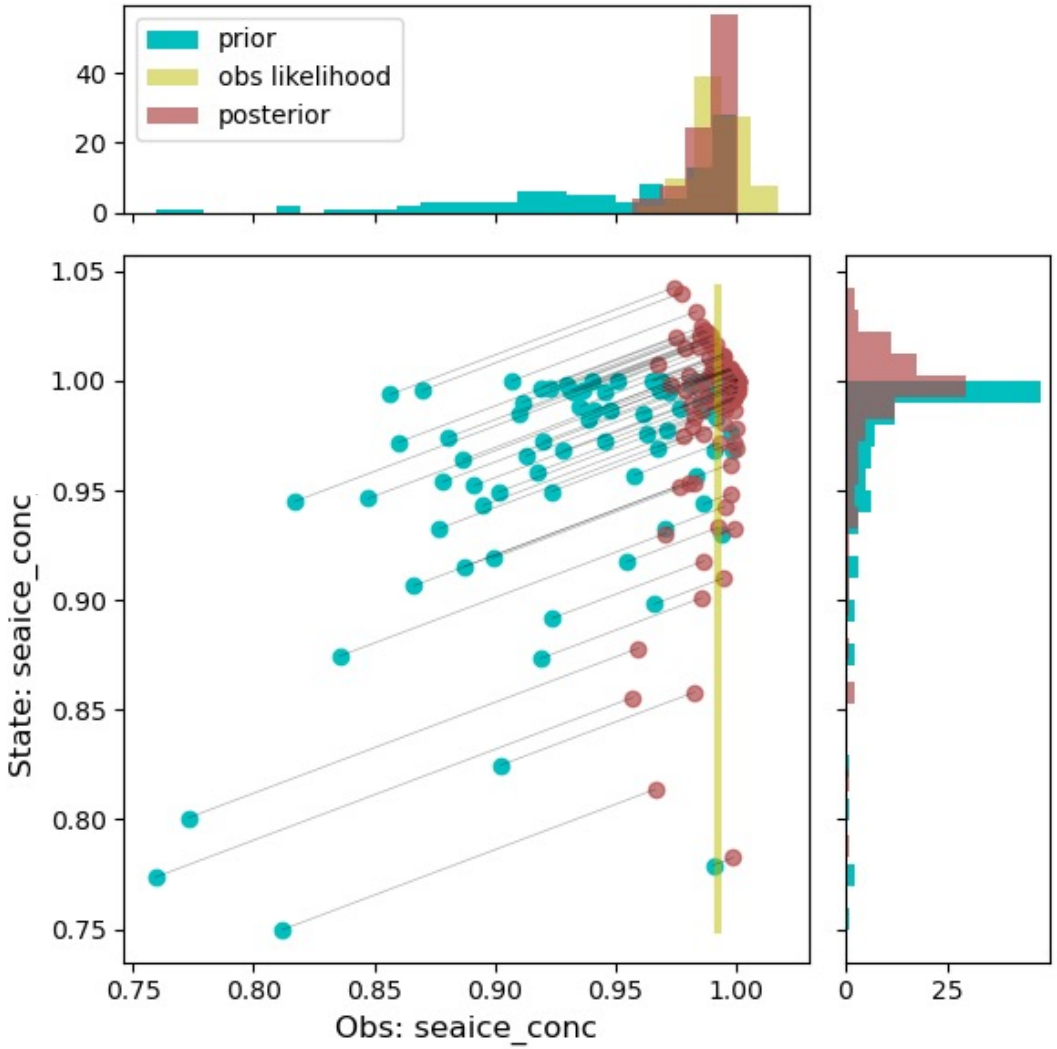
30 member



- Even with 100 members, there are still sampling noises
- Smaller ensemble size + vertical localization: cost effective?

NERSC

QCEFAssimilator as an alternative



Non-Gaussian error distribution:

- Sea ice variables
- BGC tracers

Current solution: postprocessing (fixhycom)

Quantile Conserving Ensemble Filter (Anderson 2023)

expected to better handle bounded error distribution and nonlinearity

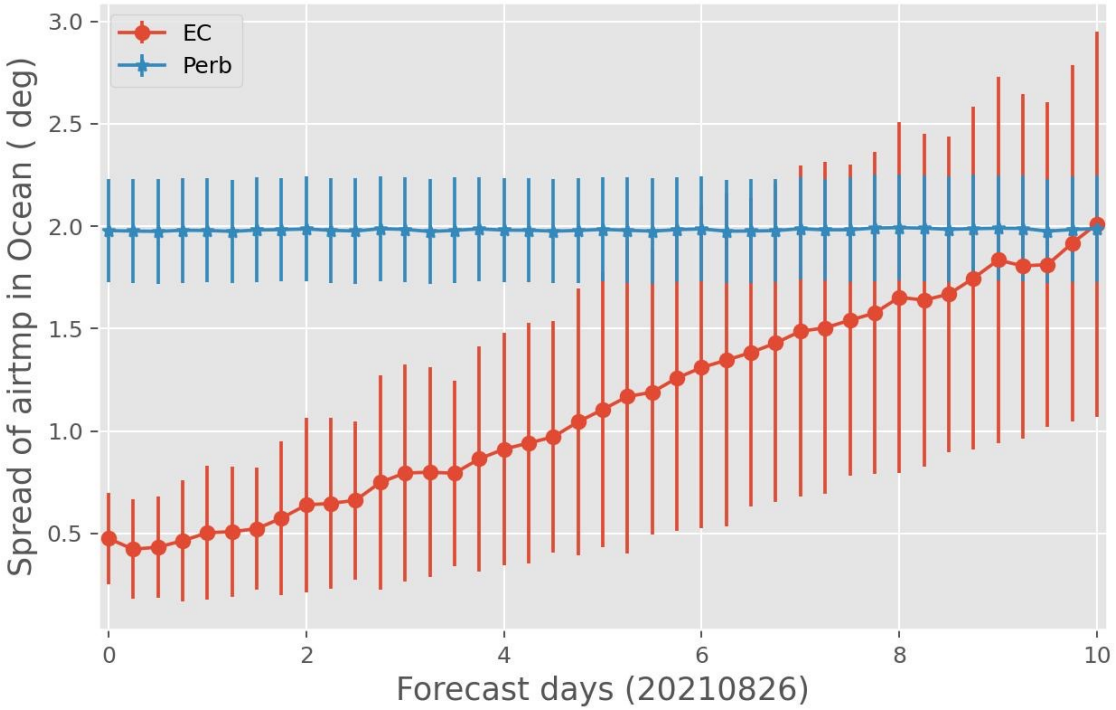
Porting QCEF subroutines to NEDAS using **f2py**

NERSC

TOPAZ perturbation scheme

Evensen 1994 approach to draw random fields: specified hcorr and variance

Real forcast errors grow in time, but our perturbations are fixed amplitude



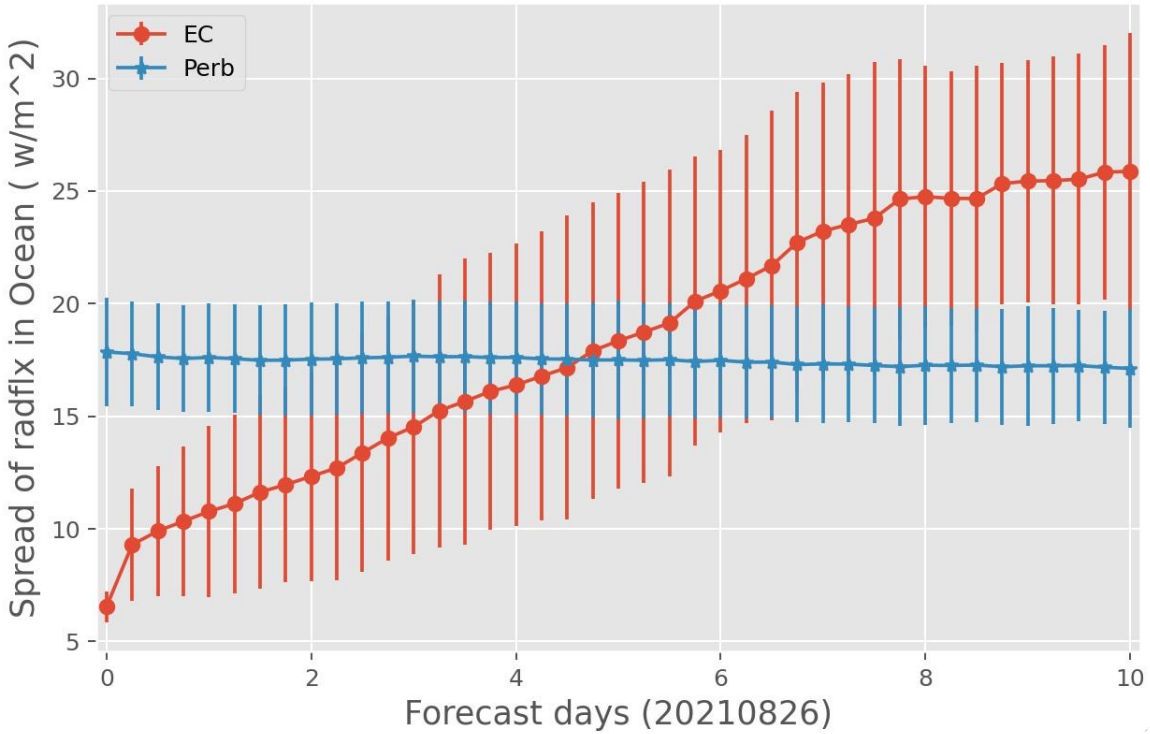
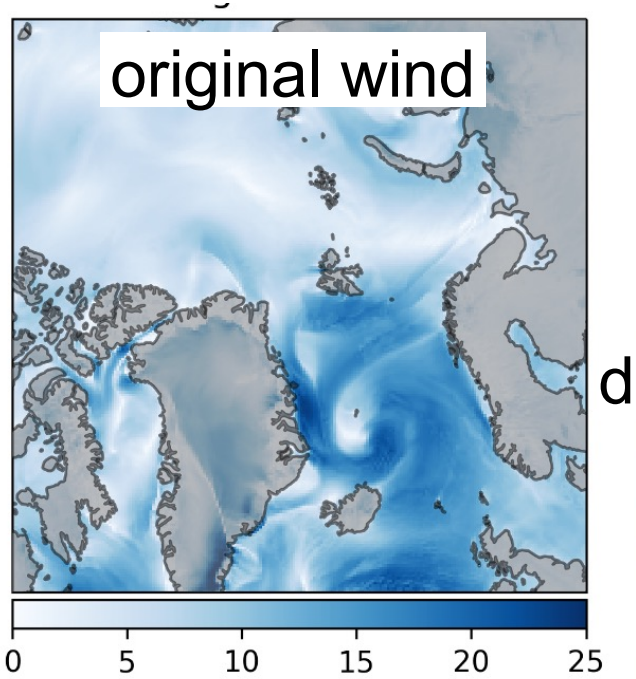


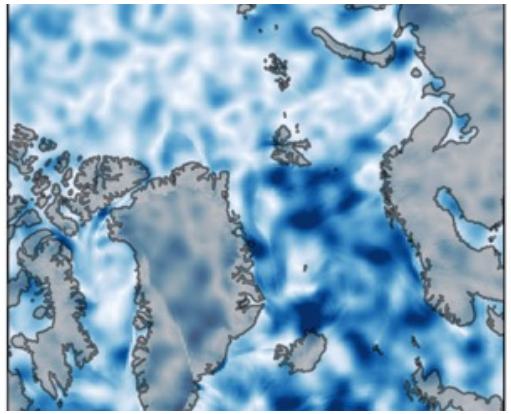
image credit: Jiping Xie

NERSC

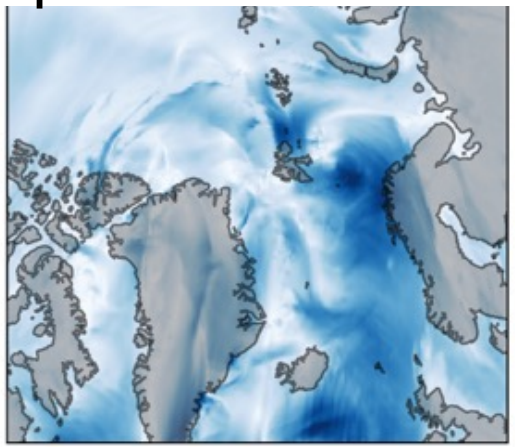
Additional perturbation schemes



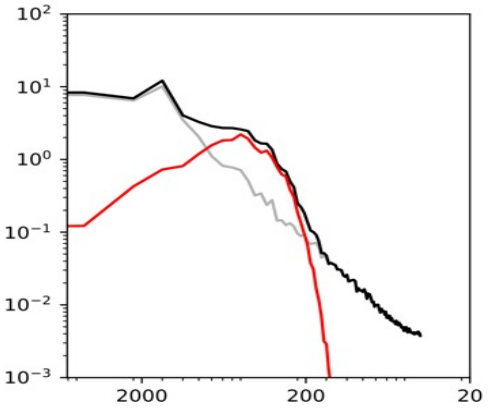
perturbed in TOPAZ

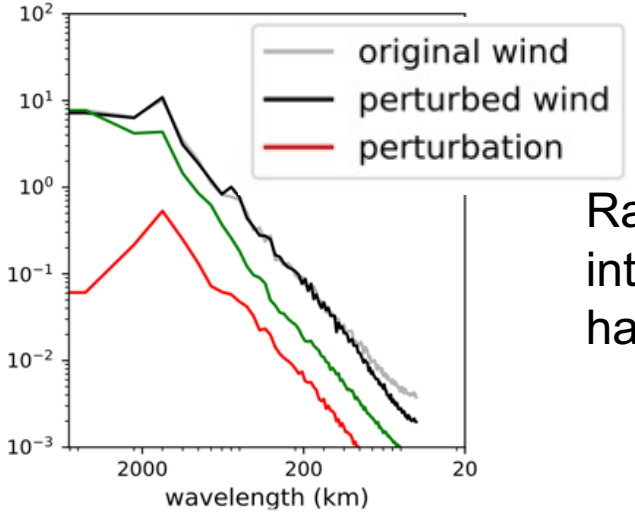


displacement scheme



energy spectrum





Current scheme:
Gaussian spectrum

Random displacement:
introduces **phase error** that has the same spectrum

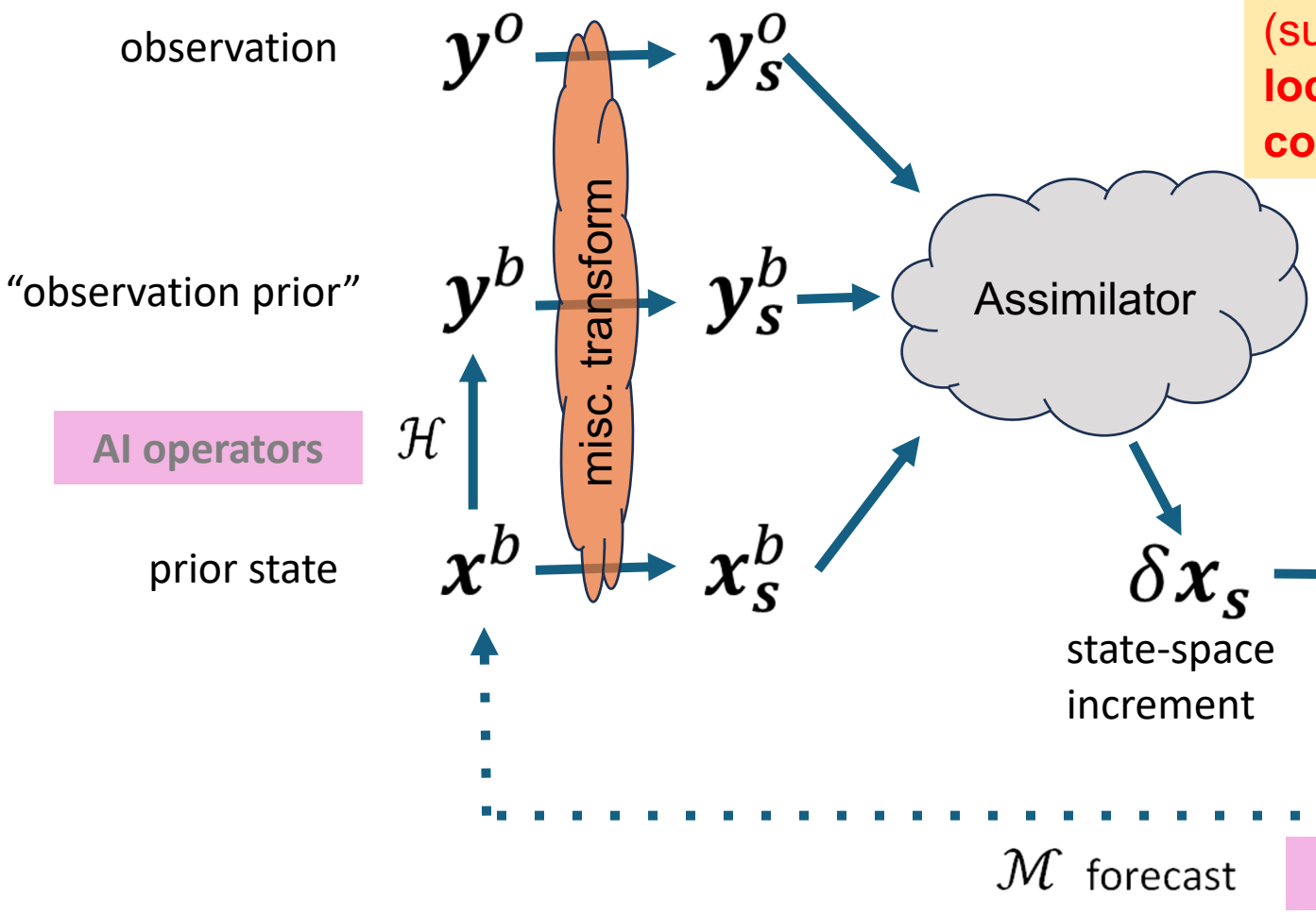
NERSC

AI enhancements (low-hanging fruits compared to end-to-end)

Misc. transform of prior state/obs
such as scale decomposition, anamorphosism…

Assimilator:
EnKF (different variants)
other non-Gaussian filters
(such as QCEF)
localization funcs…
covariance models…

AI non-Gaussian core algorithms?

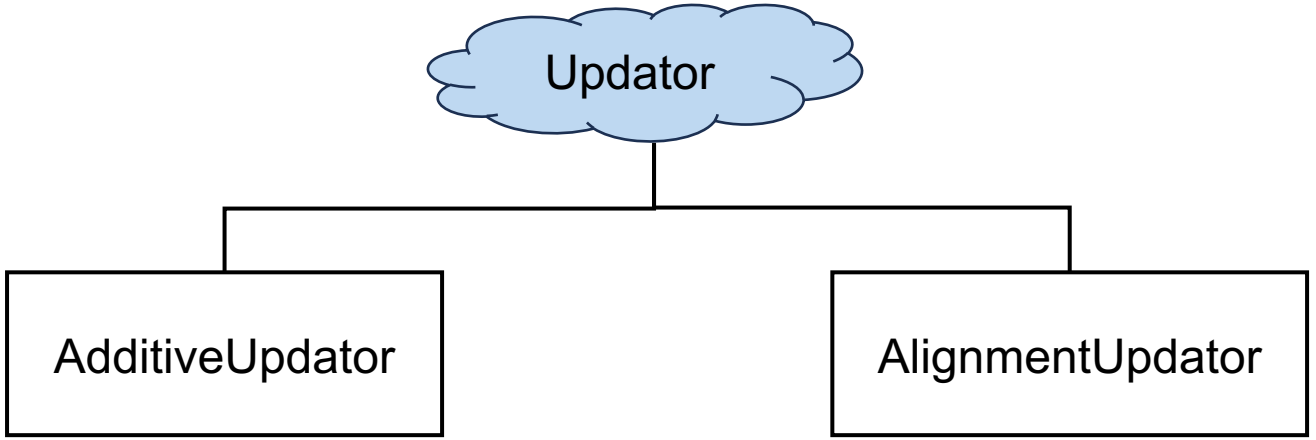


Updator:
Inverse transform
or other enhancement

AI enhancement:
super-resolution
bias correction
alignment

NERSC

AlignmentUpdator



$$\mathbf{x}^a = \mathbf{x}^b + F_s^{-1}(\delta \mathbf{x}_s)$$

"optical flow"

$$\mathbf{q}_s = \operatorname{argmin} \left\| \mathbf{x}_s^b(\mathbf{q}_s) - \mathbf{x}_s^a \right\| + w \|\nabla \mathbf{q}_s\|$$

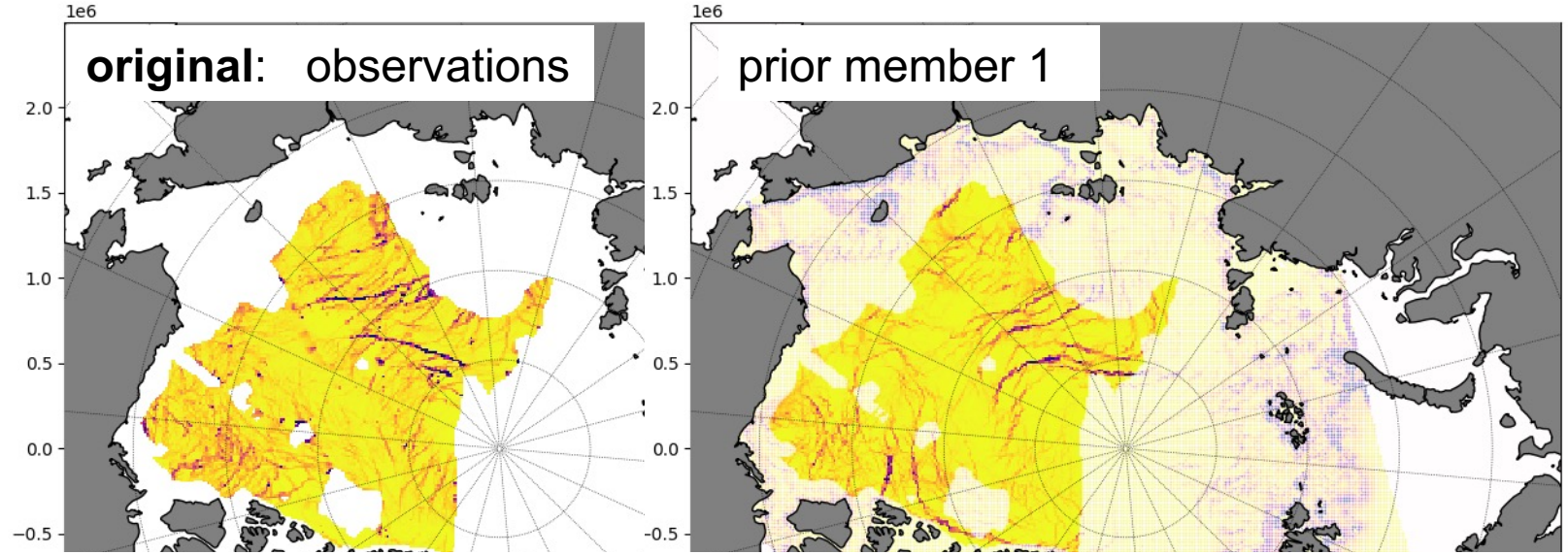
$$\mathbf{x}^a = \mathbf{x}^b(\mathbf{q}_s) + \mathbf{x}_s^a - \mathbf{x}_s^b(\mathbf{q}_s)$$

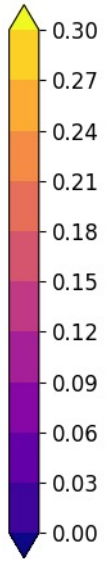
Multiscale Alignment (Ying 2019, MWR)

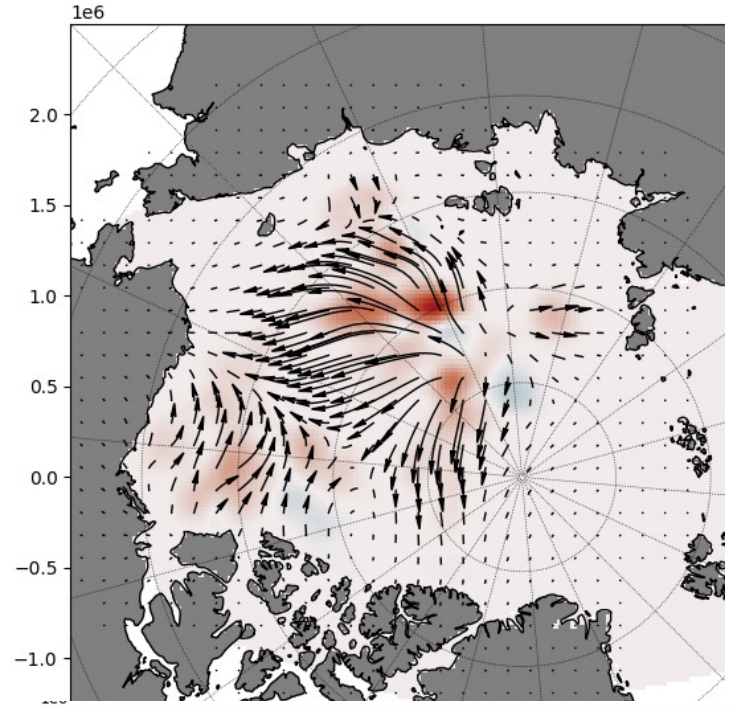
Applied to assimilation of deformation in neXtSIM
(to be coupled to HYCOM in next version TOPAZ)

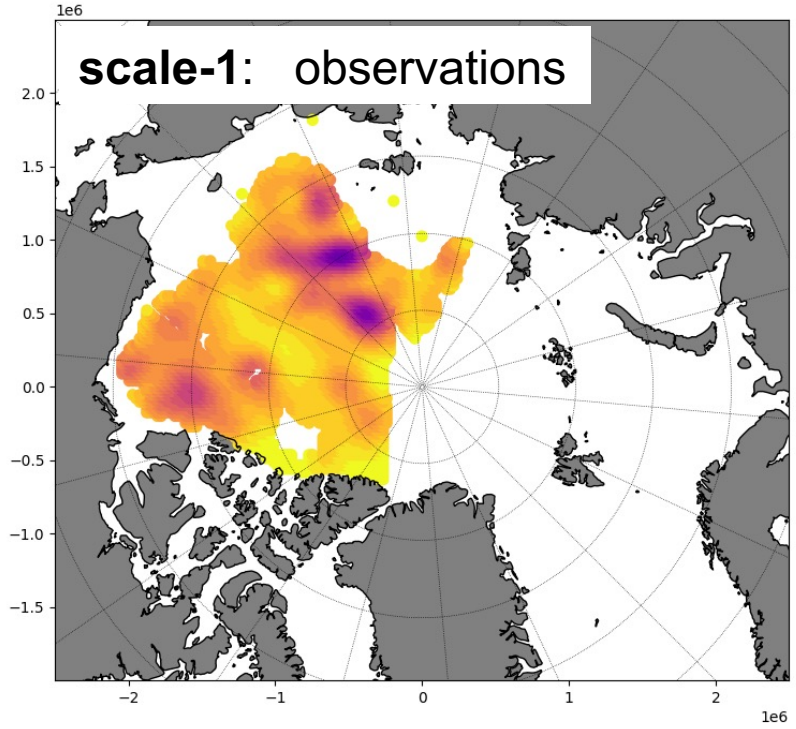
NERSC

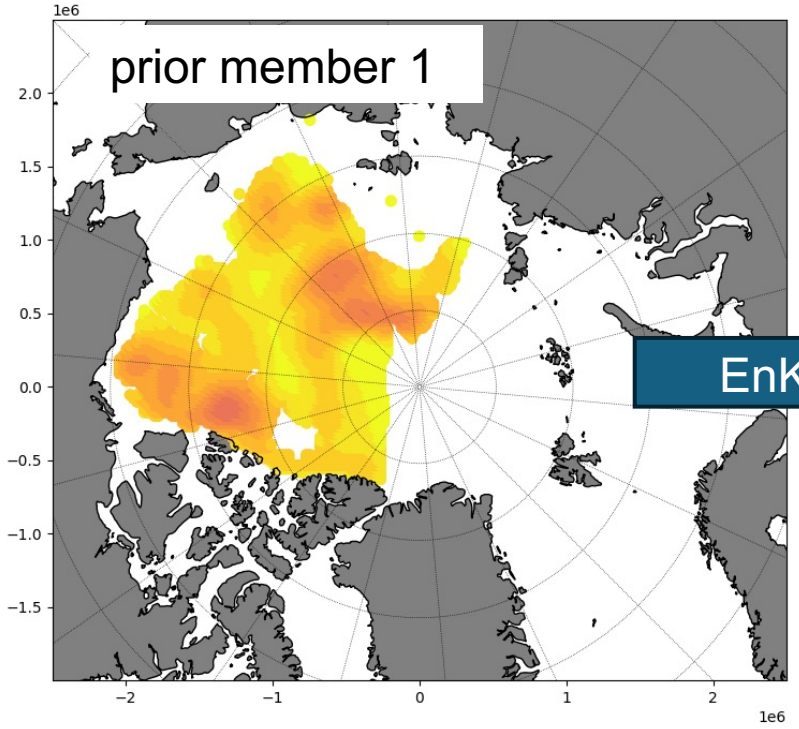
AlignmentUpdator in use for s = 1:



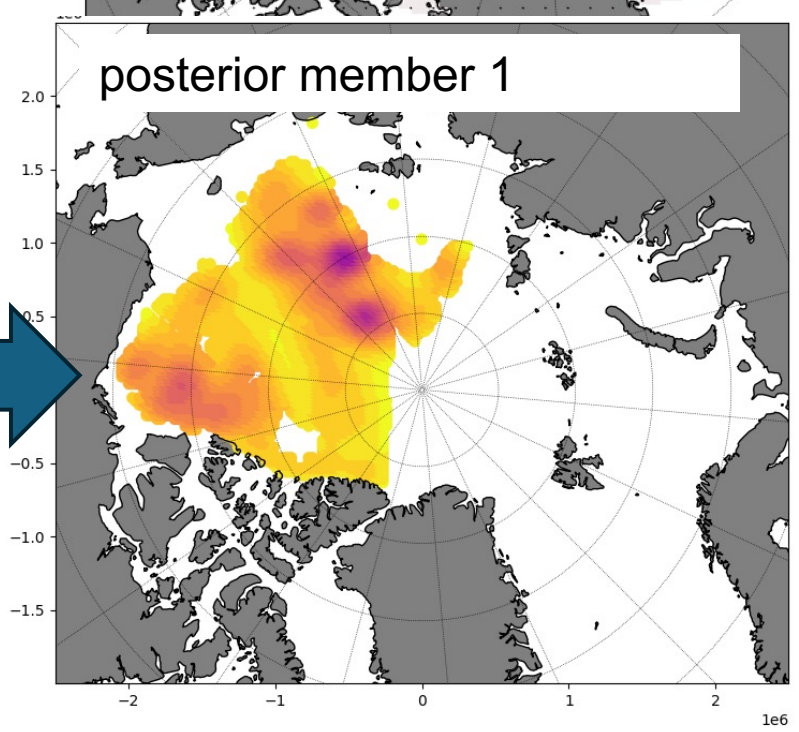




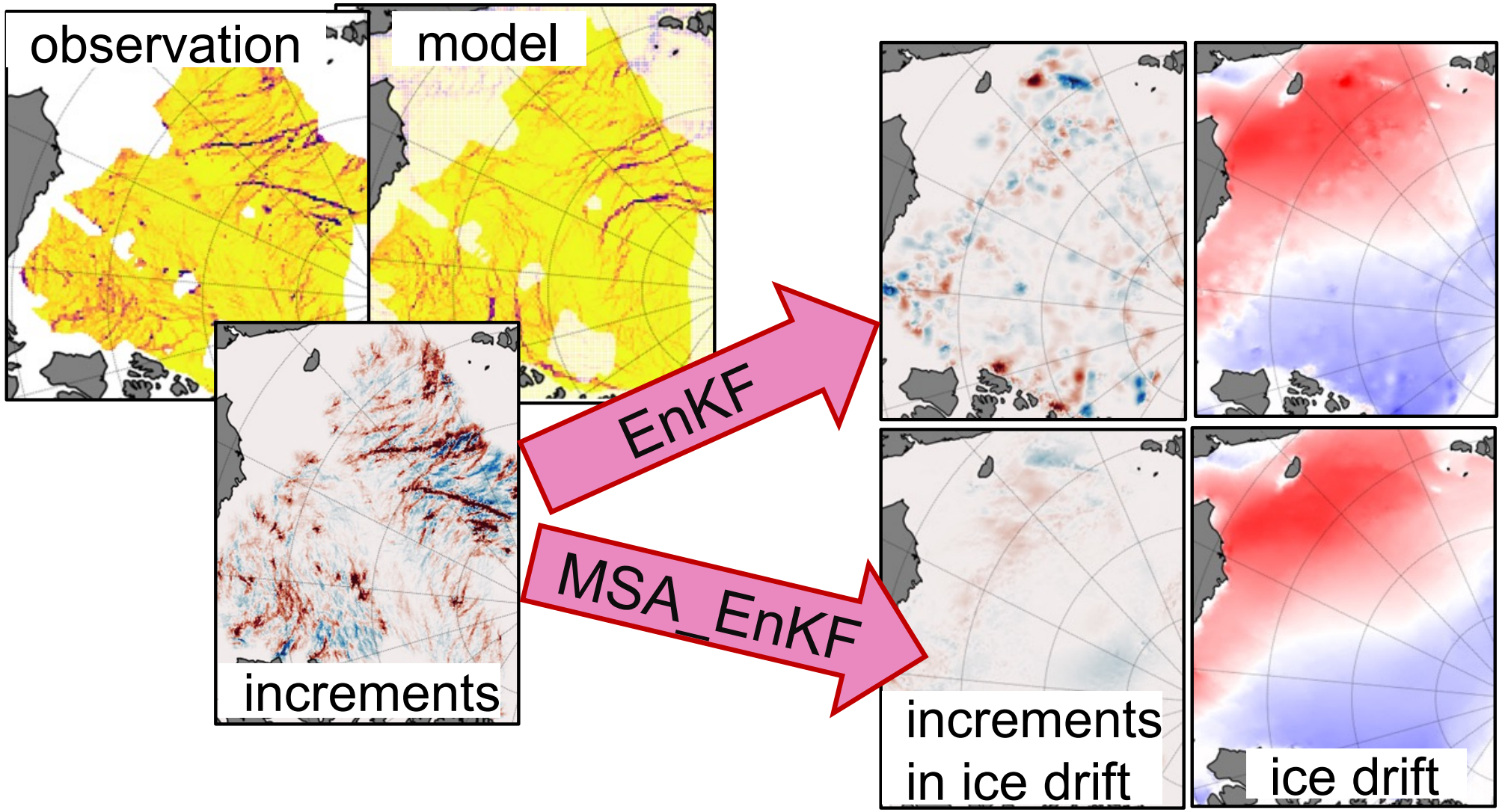




EnKF



AlignmentUpdator



NERSC

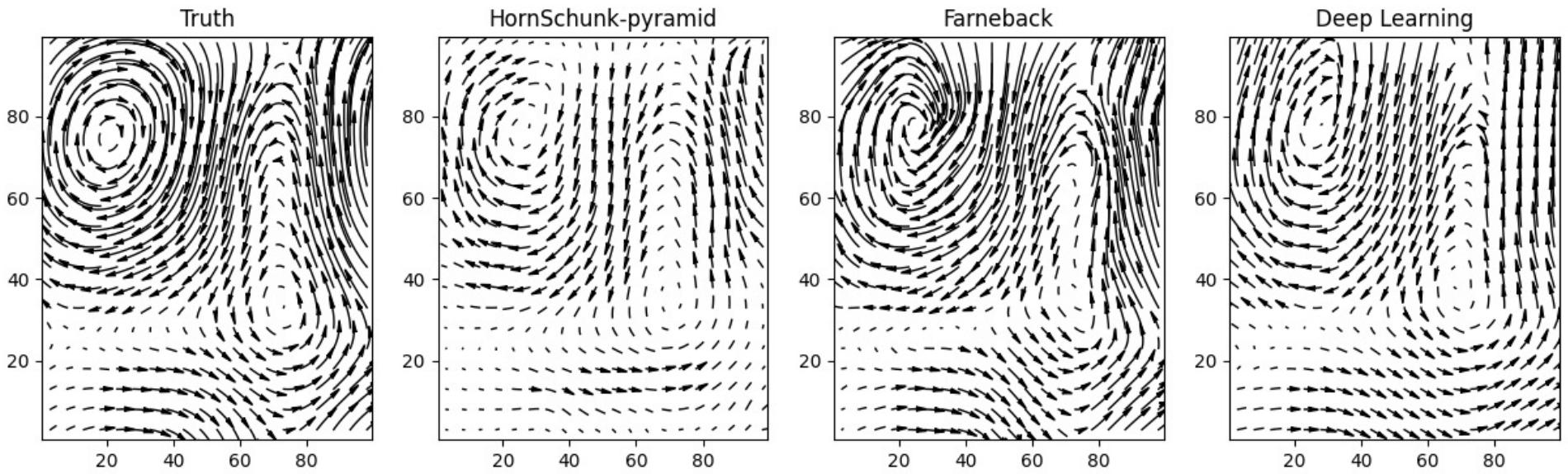
Alternative optical flow algorithms

In Ying 2019: HornSchunk with pyramid method

from OpenCV: Farneback (parametric dense flow) and DIS (based on deep learning)

```
import cv2
frame1 = ((img1+4) / 8 * 256).astype(np.uint8)
frame2 = ((img2+4) / 8 * 256).astype(np.uint8)
flow = cv2.calcOpticalFlowFarneback(frame1, frame2, None, 0.5, 3, 15, 3, 5, 1.2, 0)
```

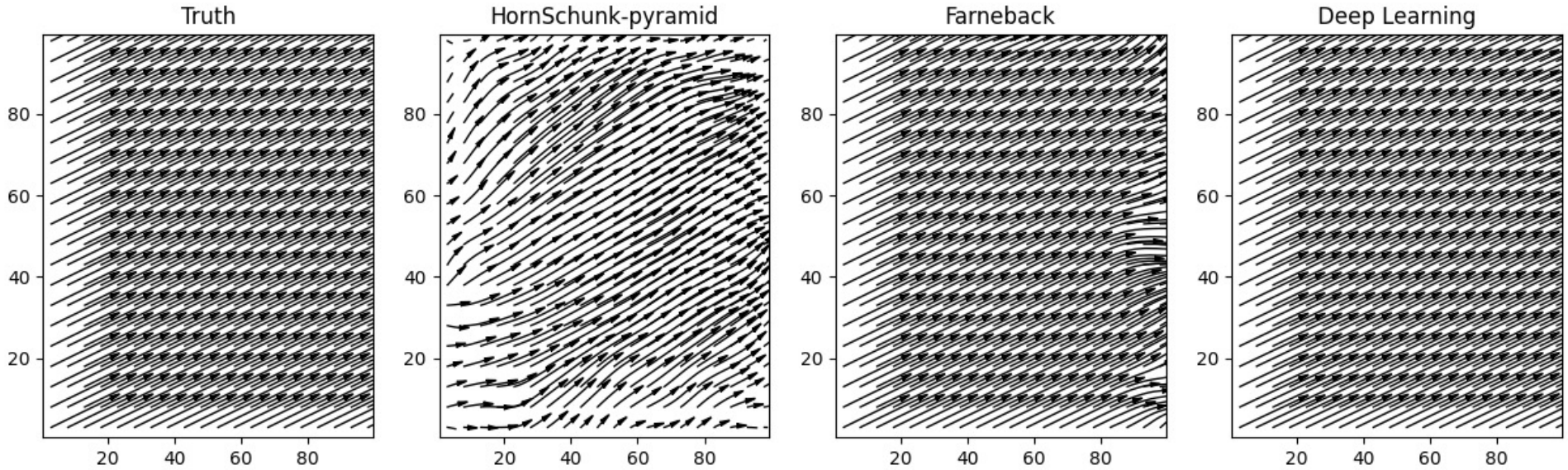
```
dis = cv2.DISOpticalFlow_create(cv2.DISOPTICAL_FLOW_PRESET_FAST)
flow = dis.calc(frame1, frame2, None)
```



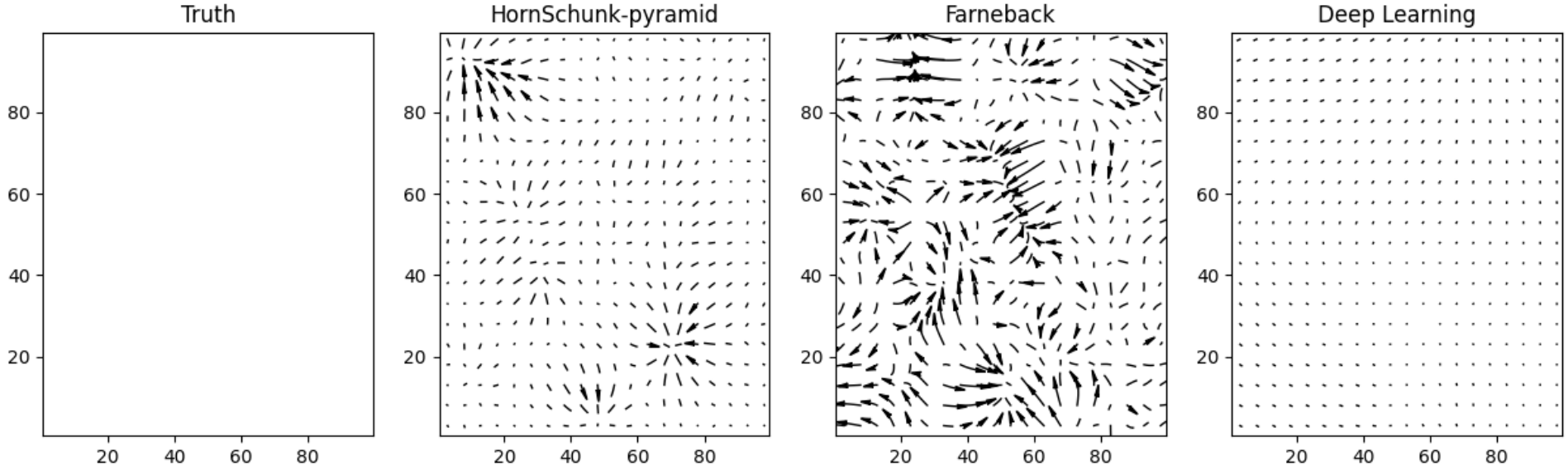
NERSC

Alternative optical flow algorithms

Constant displacement



No displacement, only amplitude changes



NERSC

Summary

Advancing TOPAZ assimilation algorithm:

- Vertical localization (finally!) and cross-variable impact factor
- Trying new perturbation schemes
- QCEF for sea ice and BGC variables (via f2py)
- Alignment technique (better alternatives from OpenCV)
- …Any suggestions?

NERSC