Highly-Scalable Algorithms for Ensemble Data Assimilation

Jeffrey Anderson, Nancy Collins, and Helen Kershaw

Institute for Mathematics Applied to Geophysics, National Center for Atmospheric Research, USA
Email: jla@ucar.edu
The Data Assimilation Research Testbed (DART)

- General purpose ensemble DA tools
- Must work with any model (global, regional, ocean, land, flu,...)
- Must work with any observation (for any model)
More DART Challenges

- Must run ‘well’ on any computer: laptop to petascale, O(10,000) cores
- Small (1 variable) to enormous (high-res coupled) models
- Arbitrary model ‘grids’, state variables
- Arbitrary observation layout: uniform or highly heterogeneous
- Any ratio of state variables to observations
- Small team supports 20+ models, many kinds of observations: Model and observation specific code must be minimal
DART Solutions

- Filter methods (adjoints are too much work for us)
- Sequential/serial ensemble filters
  
  Allows completely general localization (Lili Lei’s talk this afternoon)

\[ \Delta x_n = \alpha \hat{b} \Delta y_n , \quad n = 1, \ldots, N \]

Important for optimal filtering for general applications

- Allows more general parallelization (this talk)
Sequential Ensemble Filter

1. Use model to advance ensemble (3 members here) to time at which next observation becomes available.

Ensemble state estimate after using previous observation (analysis) vs. Ensemble state at time of next observation (prior)
2. Get prior ensemble sample of observation, $y = h(x)$, by applying forward operator $h$ to each ensemble member.

Theory: observations from instruments with uncorrelated errors can be done sequentially.
Sequential Ensemble Filter

3. Get observed value and observational error distribution from observing system.
Sequential Ensemble Filter

4. Find the increments for the prior observation ensemble (this is a scalar problem for uncorrelated observation errors).

Note: Difference between various ensemble filters is primarily in observation increment calculation.
5. Use ensemble samples of $y$ and each state variable to linearly regress observation increments onto state variable increments.

Theory: impact of observation increments on each state variable can be handled independently!
Sequential Ensemble Filter

6. Advance model to time of next observation. Usually many observations assimilated at same model time, so no need for forecast between these.
Constraints on Parallel DART Ensemble Filter

- Most models not designed with ensembles in mind
  Geophysical models often have domain decomposition with haloes
  Large subdomains -> less communication and redundant computation

- All ensemble members for a state variable or observation
  must be on the same process
  Otherwise communication costs would dominate because of repeated
  computation of mean and variance

- Transpose of data storage between forecasts and DA
  Almost all state variables must be moved to different process
  But can be done on local subsets of processes
  This must be done very efficiently on large machines
Some Simplifications for this Talk

➢ Just look at univariate horizontal model grid
  Earth system models generally have vertical columns and multiple variables at each point
  Model state is rarely distributed in the vertical

➢ Extending to three dimensional multivariate case is straightforward
Computing Forward Operators

- Forward operator $h$ is an arbitrary function of state variables
- Often ‘local’ function of state (interpolation)
- Can be complex and non-local (radio occultation ray tracing)
Computing Forward Operators

- We use MPI-2 non-blocking remote memory direct access
- One process computes $h$ for all ensemble members
- Grabs any required state variables from other processes
- Number of messages minimized if adjacent state is on same process
- Communication volume minimized if local state is on process computing $h$
Ensemble Filter for Large Geophysical Models

Bayes says ‘unrelated’ obs can be processed simultaneously!

Localization of observation impact is required for good ensemble filters

Observations that are not ‘close’ by localization are ‘unrelated’

Theory: impact of ‘unrelated’ observation increments on each state variable can be applied in any order.
Parallelizing Processing of Observations

- Find subsets of ‘unrelated’ observations
- Want minimum number of subsets
- Forward operators and observation increments can be done in parallel
- All state variables must be incremented before next subset
Parallelizing Processing of Observations

- Finding minimum number of subsets with observations that aren’t close

- This is a mutual exclusion scheduling problem
  (common in computer science)

- Equivalent to a map coloring problem:
  Find minimum number of colors for a ‘map’ of observations such that no two close observations have the same color
Solving the Map Coloring Problem

- Exact solution is NP-hard (very, very expensive)
- Cheap algorithms can give approximate answers
- Use a greedy algorithm, chooses coloring without global information
- Decreasing Greedy Mutual Exclusion (DGME)
Find number of observations close to each observation

Localization radius is 0.25
Domain is 1x1
Coloring with DGME

Color observation with most close neighbors
Coloring with DGME

None of this observation’s neighbors can have the same color
Coloring with DGME

Pick observation with most neighbors that isn’t close to the first red
Coloring with DGME

Pick observation with most neighbors that isn’t close to a red observation
Coloring with DGME

Pick observation with most neighbors that isn’t close to a red observation
Coloring with DGME

Pick observation with most neighbors that isn’t close to a red observation
Coloring with DGME

Pick observation with most neighbors that isn’t close to a red observation
Coloring with DGME

Pick observation with most neighbors that isn’t close to a red observation
Coloring with DGME

Pick observation with most neighbors that isn’t close to a red observation
Coloring with DGME

When no more can be red, repeat for next color
Coloring with DGME

When no more can be yellow, repeat for next color
Coloring with DGME

When no more can be blue, repeat for next color
Coloring with DGME

- Four colors is enough for this example
  Get four subsets of observations that can be assimilated sequentially

- Size of subsets varies, smaller as colors proceed
  This could be a load balancing concern for real problems
Randomly distributed observations (~127,000)
Location radius 0.05
Average of 1000 close neighbors

Red shows observations in a given color subset
Grey background shows 2% of total observations
Randomly distributed observations (~127,000)
  Localization radius 0.05
  Average of 1000 close neighbors

Number of observations per color nearly constant for most colors
  Load balancing not an enormous problem
For randomly distributed observations
Number of colors is mostly function of average number of close obs
This is just observation density normalized by localization
Number of colors is slightly greater than half the average number close

Number of close observations is proportional to information content

Computational cost is linear in information content
Add in simulated satellite track with 25,000 observations (~152,000 total)
Localization radius 0.05

Red shows observations in a given color subset
Grey background shows 2% of total observations
Add in simulated satellite track with 25,000 observations (~152,000 total)
Localization radius 0.05

Background observations finished after 500 colors

Then many fewer observations per color, primarily in satellite track
Load balance issues more serious
Add in two convective regions with radar observations (~202,000 total)
Localization radius 0.05

Red shows observations in a given color subset
Grey background shows 2% of total observations
Add in two convective regions with radar observations (~202,000 total)
Localization radius 0.05

Last colors only have a few observations each
These are in regions where satellite and radar overlapped
May be significant load balance issue
Preliminary Conclusions

- For common atmospheric observation distributions
  - Can process many observations in parallel
  - Number of colors is small compared to number of observations
  - This implies significant speedup

- But, what about those state variables?
Ensemble Filter for Large Geophysical Models

For each color, can do following independently in parallel:
   (Each observation can be handled by a separate process)

Compute forward operators, \( h \)  Compute observation increments

Must compute all increments for state variables before starting next color.
Updating State Variables with One Color’s Observations

- Compute regression coefficient between observation and state ensembles
- Regress observation increments onto state ensemble
- Distribution of state variables onto processes controls performance
  - Load balance, slowest process controls speed
  - Communication, send obs increments to process with state
Distribution of State Variables on $P=2^N$ Processes

- Total work per process is proportional to:
  
  \[
  \text{Sum of number of observations close to each state variable}
  \]

- Here, look at square $1000\times1000$ grid of state variables
  
  Total 1 million (real problems have vertical, multivariate, too)

- Examine 3 distributions:
  
  1. Random: Each process gets random $1/P$ of state variables
  2. Uniform: Divide domain into $P$ squares of same size
  3. Balanced: Total work for all observations approximately equal
Finding a balanced work partition of state variables

- Divide state grid points into $P$ rectangular subsets
  Minimize the maximum total work for any subset

- Optimal balanced rectangular partition problem
  Common in parallel computing applications
  NP-complete; optimal solution very, very costly

- Can use another local greedy algorithm
  Computationally cheap
  Produces a good approximate solution with high probability
Balanced Rectangular Partitions for 1000x1000 grid points

Grey dots show 2% of observations

Partition so sum of total work on each side of line is nearly equal
Balanced Rectangular Partitions for 1000x1000 grid points

Partition each rectangle so sum of total work on each side is nearly equal
Balanced Rectangular Partitions for 1000x1000 grid points

Partition each rectangle so sum of total work on each side is nearly equal
Balanced Rectangular Partitions for 1000x1000 grid points

Partition each rectangle so sum of total work on each side is nearly equal
Balanced Rectangular Partitions for 1000x1000 grid points

Partition each rectangle so sum of total work on each side is nearly equal
Balanced Rectangular Partitions for 1000x1000 grid points

Partition each rectangle so sum of total work on each side is nearly equal
Balanced Rectangular Partitions for 1000x1000 grid points

Partition each rectangle so sum of total work on each side is nearly equal
Load Balance for Different State Variable Distributions

- Two observation sets:
  - Random with ~127,000 observations
  - Random plus satellite track plus two radars with ~202,000

- Three state variable distributions:
  - Random
  - Uniform
  - Balanced

- Two processor counts:
  - $1024 = 2^{10}$
  - $8192 = 2^{13}$
Load Balance for Different State Variable Distributions

Random observation locations, 8192 processes, 1 million state variables

All processes have same amount of work for optimal balance

Difference between max work per process and mean shows poor balance

Random is best, nearly optimal
Uniform, Balanced Similar
Take about twice as long
Load Balance for Different State Variable Distributions

Random + Sat + radar observations, 1 million state variables
Random always best
Uniform better than Balanced for early observation colors
Balanced better than Uniform for later observation colors

1024 Processes

8192 Processes

NCAR
NSF
Load Balance for Different State Variable Distributions

- Random layout is good for all cases examined

- Works well even if number of state variables per process gets small

- This is what is currently implemented in DART
  Scales well to several thousand processes for large models

But, load balancing isn’t everything
Communication Cost for Different State Variable Distributions

- State variable distribution impacts communication cost
  - Total number of messages
  - Total amount of bytes moved

- Two types of communication operations here
  1. Getting state variable ensembles for forward operator
  2. Sending obs increments to state variables
  - Second dominates cost
Communication Cost for Different State Variable Distributions

Total Number of Messages for Each Observation

Balanced slightly more than Uniform

Both scale linearly with number of processes (tasks)

Random requires a message to EVERY process for each observation! Very costly.
What Matters when Selecting State Variable Distribution?

- Relative cost of computation versus communication (machine dependent)
- Relative density of state variables and observations
- Observation density (number of observations close to given observation)
- Amount of observation spatial heterogeneity

- DART will implement all three distributions discussed here
  Also tools for implementing arbitrary distributions
- More focused efforts can probably pick one
Other Optimizations we are exploring

1. More state variable partitions than processes
   - For balanced partition, make $kP$ partitions where $P$ is number of processes, $k$ is a small integer
   - A process gets partitions with both large and small numbers of observations
   - Large partitions have more work for early colors
   - Small partitions have more work for late colors
   - Mix can have better load balance thru time
Other Optimizations (II)

2. Pre-compute observations for more than one color
   - Theory allows computing $h$ for any number of obs at one time
   - Observations from later colors are updated just like state variables
   - Can ‘smooth’ out load for state increments
   - Extra computation for observations (may be small)
   - Current DART computes ALL observations at once

3. Divide observations into pieces, separate state variable partition for each
   - Do uniform observations first, then satellite, etc.
   - Increased communication to move state variables for different partitions
Making Effective Use of Coprocessors

- Many fast, cheap processors available for each process
  - GPUs
  - INTEL Phi

- Communication to coprocessors (even from local memory) is slow
  - Getting a message from off-processor can be really slow
Making Effective Use of Coprocessors

For efficient use, need lots of computation per communication

Look at number of state variable updates per received observation

Balanced, Uniform very similar

Scales very well to many processes

Optimistic about making efficient use of coprocessors

Random has much less work per communication, probably won’t work
Conclusions

- General purpose ensemble filters can scale well to many processes
- Large geophysical problems will scale easily to $O(10000)$ processes
- General purpose facility must support flexible data distribution
- Efficient use of coprocessors may be possible
- A parallel implementation simulation facility is useful

Learn more about DART at http://www.image.ucar.edu/DAReS/DART