Parallel Implementation of an Ensemble Kalman Filter

P.L. Houtekamer – BIRS 2013

Environment Canada
Environnement Canada

18-22 February 2013

credits

The EnKF benefits from contributions by many:

- Xingxiu Deng: running many experiments and operational transfers,
- Herschel Mitchell: science issues,
- Bin He: Fortran code optimization,
- Seung-Jong Baek: Fortran code modularization,
- Ervig Lapalme: script development,
- Jeffrey Blezius: Fortran code unification with En-Var group,
- Normand Gagnon: model development and user support,
- management: unwavering support.

overview

1 Introduction

- parallel computers

2 The EnKF algorithm

- Monte Carlo approach to data assimilation
- sequential algorithm
- time interpolation
- localization

3 Scaling

- computation of analyses
- strong scaling
- weak scaling
- application towards better results

4 Discussion

- code complexity and evolution
- scientific challenges

introduction

Over recent years, we have been increasing all resolution related parameters of the Canadian operational global EnKF.

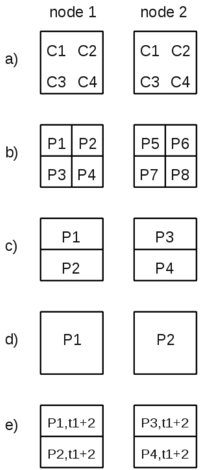
Implementation		N_{lon}	N_{lat}	N_{lev}	N_{ens}	N_{obs}	cost
January	2005	300	150	28	96	100 000	1
August	2011	400	200	58	192	300 000	22
February	2013	600	300	74	192	700 000	148

As a - not so bad - assumption we take:

$$cost = O(N_{lon} \times N_{lat} \times N_{lev} \times N_{ens} \times N_{obs})$$

The substantial increase in resolution and cost has been made possible by an equally substantial increase of the capacity of the super computer clusters at our center.

Example computer cluster with 2 nodes, each having 4 cores



node 1 | node 2

a) C1 C2 C3 C4 | C1 C2 C3 C4

b) P1 P2 P3 P4 | P5 P6 P7 P8

c) P1 P2 | P3 P4

d) P1 | P2

e) P1,t1+2 P2,t1+2 | P3,t1+2 P4,t1+2

a) The computer cluster has 2 nodes each with 4 cores sharing memory.
b) Eight processes run without sharing memory (pure MPI model).
c) Four processes run without sharing memory. Each process uses two physical cores.
d) One process runs on each code. A process uses all the memory available on its node.
e) Each process consists of two software threads which can share memory (hybrid MPI + OpenMP model).

Parallelization using the MPI Message Passing Interface

At the largest scale, parallelization on the computer cluster uses the MPI message passing interface. Different copies of a program runs simultaneously on the cluster to solve different parts of the same numerical problem. When necessary, they communicate using messages sent via the MPI protocol. Messages are sent between processes which may or may not run on the same node.

Just using MPI, it is possible for a program to make use of the entire computer cluster. For this to be efficient:

1. it must be possible to split the problem into a large number ($O(1000)$) of quasi-independent sub-problems.
2. each sub-problem should (have enough memory to) run on just one core.

parallelization using OpenMP

Within a software process, running on part of a node, one can have multiple software threads sharing memory.
Shared-memory parallelization, using OpenMP, is often at the level of loops in Fortran code as illustrated in an example from the book by Chandra et al.

```
!$omp parallel do private(j,x,y)
   do i=1,m
     do j=1,n
       x=i/real(m)
       y=j/real(n)
       depth(i,j)=mandel_val(x,y,maxiter)
     enddo
   enddo
!$omp end parallel do
```

hybrid use of MPI and OpenMP

Historically, the Canadian EnKF only used MPI for parallelization. As we are moving towards more challenging problems, with more ensemble members, more observations and more model coordinates, individual processes require increasing amounts of memory. For some problems, just to provide enough memory to a process, *we need to run with less MPI-processes per computer node.*

To still make use of the available software threads, we recently (in 2012) started using OpenMP in addition to MPI. For small problems, when available memory is not an issue, the hybrid use of OpenMP and MPI is now almost as efficient as the pure MPI application.

The POWER7 clusters

Since December 2011, our center has access to 2 IBM POWER7 clusters. Operations run with priority on one cluster. The other cluster, identical in principle, is exclusively for R&D.

Each of these clusters, has approximately 250 compute nodes and each of these nodes has 32 cores. A single user program could use an entire cluster in several ways:

- 8000×1: run 8000 MPI-processes each having only one thread,
- 4000×2: run 4000 MPI-processes each having 2 OpenMP threads,
- ...
- 250×32: run 250 MPI-processes each having 32 OpenMP threads.

trends

Originally, our prototype EnKF could run on a computer with only one process (1 CPU). This program, which was coded in Fortran77, used neither MPI nor OpenMP.

With the arrival of NEC vector processing machines, we introduced parallelization with MPI to use 2-4 vector processors.

With the arrival of the IBM P7 clusters, to handle bigger problems, we had to also introduce OpenMP. The hybrid MPI and OpenMP parallelization matches the structure with nodes and cores of the POWER7.

How well a program exploits the available computer resources translates directly into the size of problem that can be handled and thus also into the quality of the results.

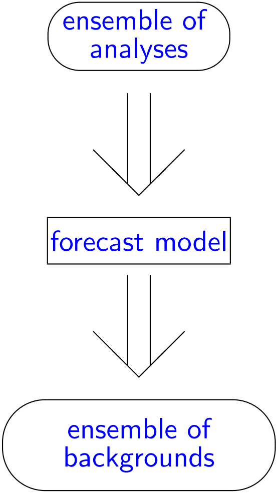
Defining characteristics of the Canadian EnKF

Gradually, the Canadian EnKF evolved into a system having:

1. Monte Carlo method (1998),
2. cross-validation (1998),
3. additive model error (2000),
4. localization using a Schur product (2001),
5. sequential algorithm (2001),
6. extended state-vector approach (2005).

We do not claim there is no better algorithm possible, but we think we can make substantial progress staying in the current framework.

a Monte Carlo ensemble of model runs

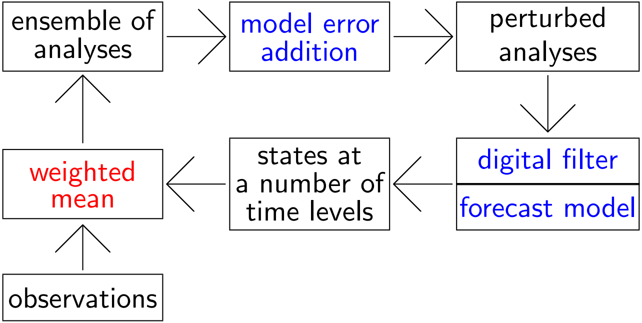


At a technical level, the application of the Monte Carlo method is trivial. One just has to rerun a system a number of times.

In this example, the forecast model is considered to be a black box. It is provided with an input analysis and provides a background trajectory as an output.

If the model itself is uncertain, this uncertainty will have to be sampled as well.

EnKF flowchart



For each of $N_{ens} = 192$ ensemble members, we go through a 6h data-assimilation cycle. All information comes together in the computation of the weighted mean, where the weights are computed using the ensemble of N_{ens} background fields.

the Kalman gain

The weighted mean is obtained using the Kalman Gain matrix:

$$\mathrm{K}_e = \mathrm{P}_e^f \mathrm{H}^T (\mathrm{H}\mathrm{P}_e^f \mathrm{H}^T + \mathrm{R})^{-1}$$

The difficulty here is that the analysis increments are computed using the input ensemble of backgrounds. This ensemble of backgrounds has also been used to obtain P_e. Thus we first tune the analysis using an ensemble and subsequently this same ensemble is used to test the quality of analysis.

This invalid application of the Monte Carlo method is also known as “inbreeding”.

cross validation

In the operational EnKF, we use 4-fold cross-validation to obtain the analysis ensemble:

1. The ensemble of 192 background fields is split into 4 sub ensembles of 48 members.
2. To assimilate data into sub ensemble k, $k = 1, \cdots, 4$, the Kalman gain is computed from the backgrounds of the 3 other sub ensembles.
3. The 4 sub ensembles are combined into a 192-member analysis ensemble.

For a subsequent operational delivery, we consider moving to 8-fold cross validation. Various computations involving sub ensembles have been parallelized using OpenMP.

issues with the random inputs

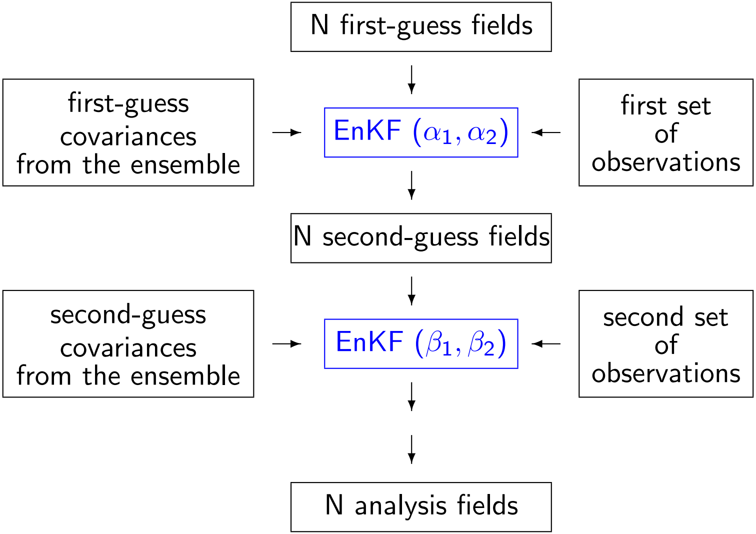
Currently, the addition of large-amplitude random fields is of critical importance to our system. We do not know what error sources correspond to these fields.

To permit a reduction of their amplitude, we likely need to sample:

1. systematic errors in the observational network,
2. model physics problems near the surface.

The EnKF system benefits from the continuous progress in all aspects of NWP. With the steady reduction of systematic errors, the natural growth of well-simulated random errors tends to become more important.

sequential algorithm



N first-guess fields

first-guess covariances from the ensemble

EnKF (α_1, α_2)

first set of observations

N second-guess fields

second-guess covariances from the ensemble

EnKF (β_1, β_2)

second set of observations

N analysis fields

sequential algorithm

The use of the sequential algorithm permits us to deal with the inverse in:

$$\mathrm{K}_e = \mathrm{P}_e^f \mathrm{H}^T (\mathrm{H}\mathrm{P}_e^f \mathrm{H}^T + \mathrm{R})^{-1}$$

using the direct Cholesky decomposition method. (The cost of the matrix inversion scales as N_{obs}^3).

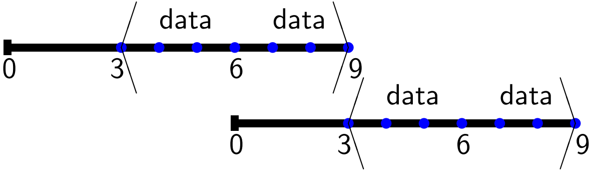
However:

- observations in different batches must have independent errors,
- when we return in the same area for a second batch the ensembles are no longer independent.

In view of these fundamental problems, Herschel Mitchell is investigating the use of a variational solution algorithm.

time interpolation

The EnKF assimilates all data in a 6-h window at the appropriate time (Hunt et al. 2004, Tellus A), as is also done in 4D variational algorithms:



For the time interpolation in the forward operator H, we need the model state at $t = 3\text{h}, 4\text{h}, 5\text{h}, 6\text{h}, 7\text{h}, 8\text{h}$ and 9h. Only the analysis at the central time $t = 6\text{h}$ is used to start the subsequent integration.

naive implementation

The cost of the data assimilation step is dominated by operations involving the matrix $\mathrm{P}_e^f \mathrm{H}^T$. These have a cost:

$$cost = O(N_{model} * N_{timelevels} * N_{obs} * N_{ens}).$$

Using $N_{timelevels} = 7$ leads to a seven-fold increase of the cost!

In the sequential algorithm, observations are assimilated one batch at a time. We need to keep track of the “evolving” trajectories because the forward operator H first interpolates the model state to the time of the observation.

expansion of the state vector

The vector Hx of interpolated observations can be added to the state vector (Tarantola 1987; Anderson, MWR, 2001), which then becomes the expanded state-vector (x, Hx).

Since H is precomputed, it is sufficient to have as state vector $(x(t = 6\text{h}), Hx(t = t_{obs}))$. The information about the temporal evolution is in the evolving correlations between $x(t = 6\text{h})$ and $Hx(t = t_{obs})$.

This revised algorithm has cost:

$$cost = O((N_{model} + N_{obs}) * N_{obs} * N_{ens}) \approx O(N_{model} * N_{obs} * N_{ens})$$

localization

Localization needs to be used - almost always - in an Ensemble Kalman Filter (EnKF) due to restrictions on the size of the ensembles. In fact, *localization is the key technique which makes the ensemble approximation to the Kalman filter computationally feasible*.

Example: with $N_{model} = 50\ 000\ 000$ in an Extended Kalman filter, we need N_{model} integrations of the tangent linear model. If we can solve the same problem with an EnKF with $N_{ens} = 250$, we have a cost reduction factor of 200 000!

How localization is best applied depends on aspects of the model dynamics and the observational network. *A reasonable choice often leads to a substantial improvement in performance.*

horizontal and vertical localization

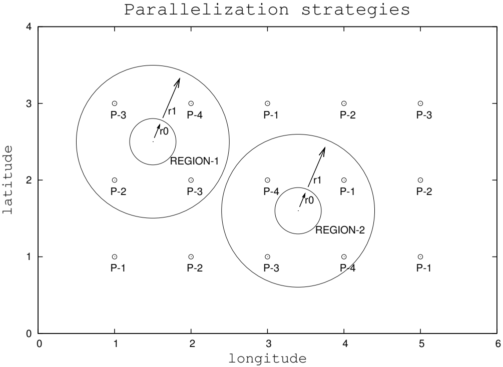
To reflect the increase of length scales with height, the horizontal localizing function $\rho_H(z)$ in the EnKF changes with height z:

$$K = \left[\rho_V \circ \rho_H(z) \circ (P^f H^T)\right] \left[\rho_V \circ \rho_H(z) \circ (HP^f H^T) + R\right]^{-1}$$

Localization in EnKF	
layer (hPa)	length (km)
2-14	3000
14-100	2800
100-400	2500
400-1050	2100

The vertical localization is in two units of $\ln(P)$.

localization with a compact function



The compact support of the localization is exploited to update the state vector only in the vicinity of a group of observations.

In one step of the sequential algorithm, we can assimilate observations from several independent areas on the analysis grid in parallel.

cost in a data-assimilation cycle

A data-assimilation cycle consists of two main steps:

1. perform an ensemble of short integrations with the forecast model,
2. use observations to obtain analysis increments.

With current parameters, **the model integration step is 7 times more expensive than the analysis step.** However, the ensemble of independent model integrations is **embarrassingly parallel and not time critical.**

How to parallelize **the analysis is non-trivial and time critical**. It is the focus of this section.

preparation of observations for the EnKF

Observations, for the EnKF, have been preprocessed by our center's deterministic high-resolution (4d-Var) data-assimilation system. We benefit from:

1. data thinning of, in particular, satellite data to about 150 km,
2. bias correction of the observations,
3. quality control with respect to a high-resolution deterministic background field,
4. a further selection of observations to include only those to be assimilated by the EnKF. The EnKF system does not use AIRS and IASI observations.

EnKF observation pre-processing

In the EnKF itself, we have 4 observation pre-processing steps.

1. Decode the observation file: 10 seconds.
   There would be some potential for parallelization.
2. Evaluate the forward operator: 2 minutes.
   The problem is almost independent for different members and scales well.
3. Background check: 16 seconds.
   The software is sequential. It could be made parallel.
4. Sort for sequential processing: 20 seconds.
   The software is sequential and could be made parallel.

This software mainly evolves as a consequence of code unification with the En-Var group. It is not discussed further.

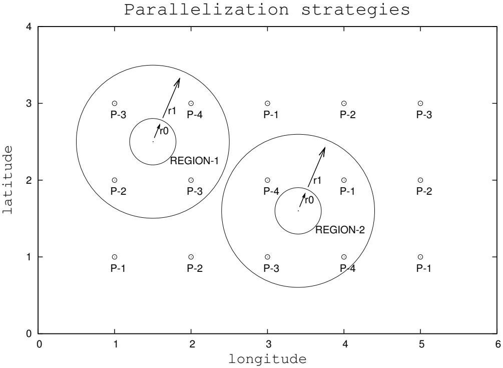
cost of analysis components

The cost of the analysis is shown for a case with 288 cores.

description of routine	seconds	percentage
update state vector	543	80.4 %
communicate observations	28	4.1 %
Cholesky decomposition	26	3.9 %
communicate H(x)	20	3.0 %
perturb observations	13	1.9 %
update extended part of state vector	10	1.5 %
write analyses	10	1.5 %
read trial fields	8	1.2 %
wall clock	675	100 %

The two routines in blue parallelize well. One nicely optimized routine takes more than 80 % of the cost. Routines in red have yet to benefit from optimization.

updating the state vector in parallel



Subsequent grid points are assigned to subsequent processes in a round-robin manner.

Since the number of grid points is large, it is easy to parallelize operations involving PH^T.

Amdahl's law

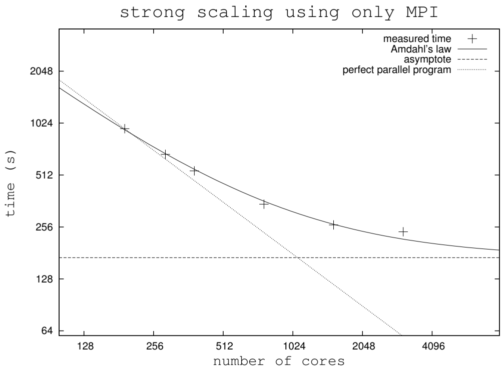
Given a sequential fraction s and N CPUs, the application speedup S is:

$$S = \frac{1}{s + \frac{1-s}{N}} \qquad \text{(Amdahl's law)}$$

In the example above, with 81.9 % parallel, we have $s \approx 0.18$. The maximum speedup, for $N \to \infty$ will be 5.5.

This relation is verified with a *strong scaling test.*

strong scaling results



In a strong scaling experiment, one tries to run a given problem faster using a bigger fraction of the cluster.

The current EnKF code does not scale well beyond say 1000 cores. At that stage, however, the analysis completes in about 5 minutes.

weak scaling

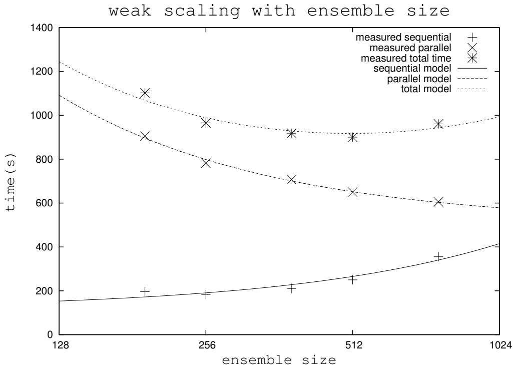
Good strong scaling is difficult because a very high fraction of the code needs to be parallelized. In practice, once we have reasonably fast execution, we are more likely to want to increase the problem size to obtain better results.

Currently, 80 % of the cost goes to updating the state vector. The corresponding operations involve the matrix PH^T. From an operation count we have:

$$\text{Cost} = N_{ens} \times N_{model} \times N_{obs}.$$

In weak scaling experiments, the number of nodes is proportional to the computed cost. Ideally, the execution time (secs) remains constant when the problem is made bigger.

weak scaling with ensemble size



Here the number of cores equals the ensemble size.

In a weak scaling experiment, one tries to run a bigger problem in the same time on a bigger fraction of the cluster.

The execution time stays about the same when more members are used. Unfortunately the fraction associated with the parallel problem decreases.

weak scaling with the number of grid points

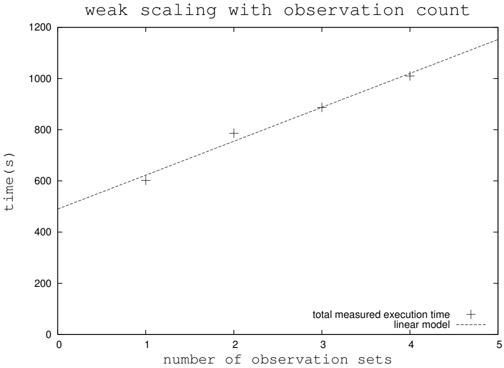
A weak scaling experiment has been performed with

- a 600 × 300 horizontal grid (as at operations since Feb. 13 2013) and
- a 800 × 400 horizontal grid (in preparation for a delivery in 2013).

The weak scaling was nearly perfect (not shown). Consequently, using more nodes we can move to higher resolution grids and maintain our current execution times for the analysis.

For a 1200 × 600 grid, one of our programs did not have enough memory to run. This has yet to be investigated.

weak scaling with the number of observations



Here, we use 384 cores per observation set.

The cost of most of the unoptimized sequential code depends on the observation count. The execution time almost doubles when the number of observations is increased with a factor of 4.

summary of scaling results

- Using a large fraction of an IBM P7 cluster, the operational EnKF could run in 4 minutes.
- A larger fraction could also be used towards having more members or bigger grids.
- Before we increase the observation count by a substantial factor (> 5), we may need to revisit the code and/or the algorithm.
- *Since we can run bigger problems, it is interesting to investigate how much quality we gain with that.*

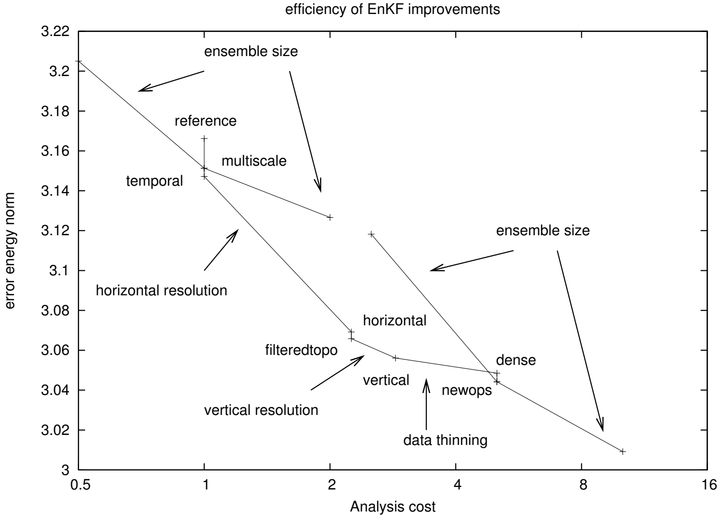
relationship between numerical cost and meteorological quality

Recently (February 13 2013), we had an upgrade to the Global Ensemble Prediction System (GEPS) and EnKF.

EnKF parameters					
Date	N_{lon}	N_{lat}	N_{lev}	timestep	N_{obs}
August 2011	400	200	58	0:30	300 000
February 2013	600	300	74	0:20	700 000

This upgrade was closely related to the migration from a pair of IBM P5 clusters (2000 cores each) to a pair of IBM P7 clusters (8000 cores each). The impact of the individual resolution changes on analysis quality will be shown.

The 13 February 2013 EnKF upgrade From "reference" to "newops"



Link with other projects

The EnKF system originated as a stand-alone system with only inputs from the global assimilation system.

Soon, we could be two-way coupled to:

1. an EnKF system for surface fields,
2. an En-Var system for the deterministic analysis,

We could be providing input to:

1. a regional EnKF (itself possibly linked to a regional En-Var),
2. a regional EPS,
3. the global EPS.

code history

Currently our center has data-assimilation software for:

1. the EnKF (which was developed from scratch and with input from the global 3D-Var),
2. the En-Var (which evolved from the global 4D-Var),
3. the regional 4D-Var (which evolved from the global 4D-Var).

Some of the code has a long history (10-20 years). Some of the original developers have now left our center or could retire soon.

New developments, like changing to a new version of the RTTOV operator, have to be introduced into the different coding systems. There is some duplication of work.

code evolution

Our center is putting a major effort into:

- the transition towards a unified job-sequencer and common scripting standards. The transition is almost completed.
- the transition towards the use of shared Fortran-90 modules. We recently managed to unify the handling of observational data between the En-Var and EnKF group.

The full unification, revision and modernization of all data-assimilation operators is a multi-year continuing effort.

scientific challenges

In the coming years, we can likely benefit from having increasingly powerful computer clusters.

At the same time, we would to like to make progress on some challenging data-assimilation issues:

1. we need to reduce the amplitude of the “model error” component,
2. we need to deal with correlated observation errors,
3. we need to reduce the model spin-up and shorten the data-assimilation window.

Thank you