Computational Efficiency

P.L. Houtekamer – 6th EnKF workshop

Environment Canada Environnement Canada

18-22 May 2014

overview

introduction

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

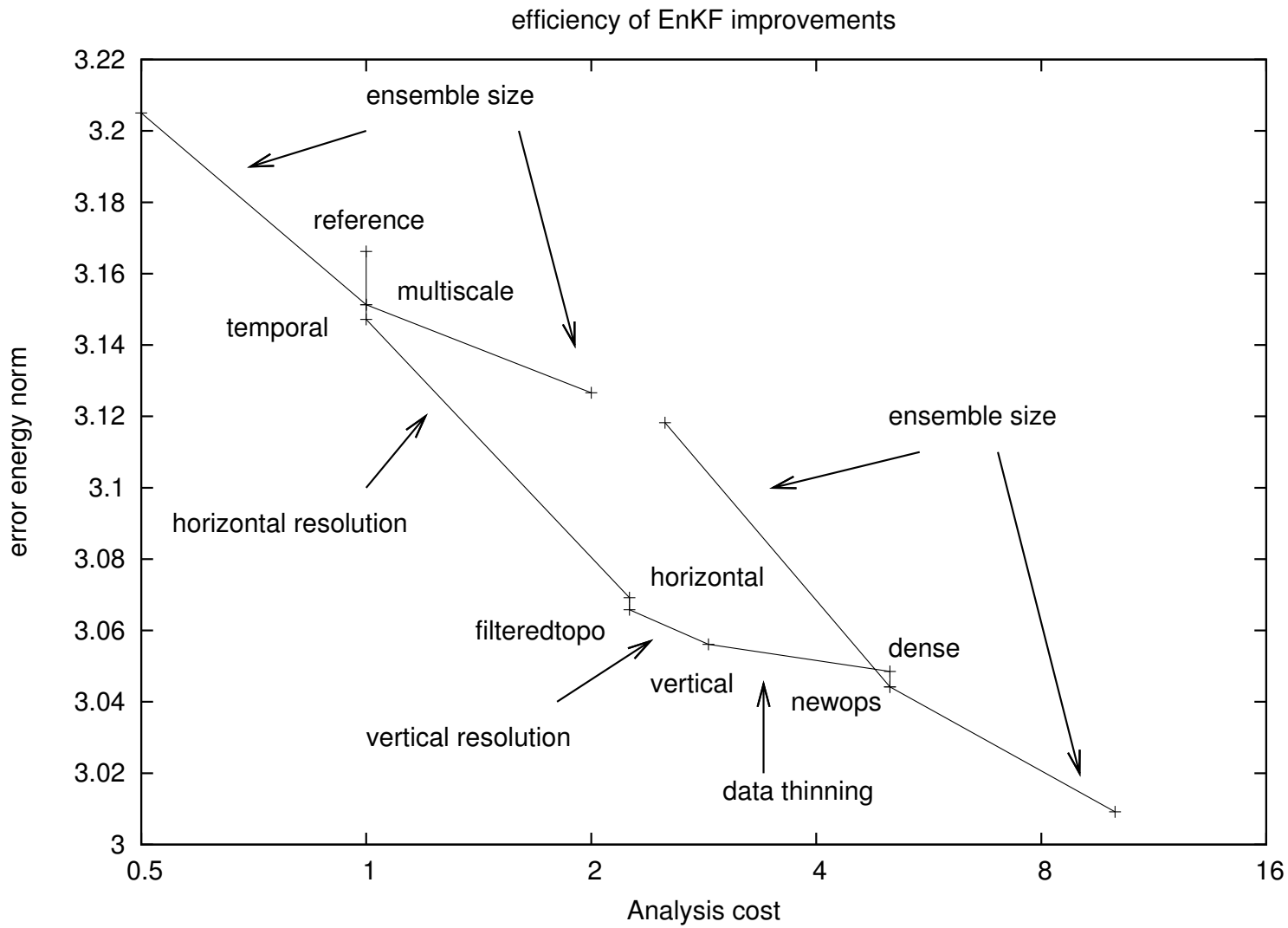
Implementation		N_{lon}	N_{lat}	N_{lev}	N_{ens}	N_{obs}	cost
January	2005	300	150	28	96	100 000	1
August	2011	400	200	58	192	300 000	22
February	2013	600	300	74	192	700 000	148
Autumn	2014	800	400	74	256	700 000	350

As a - not so bad - assumption we take:

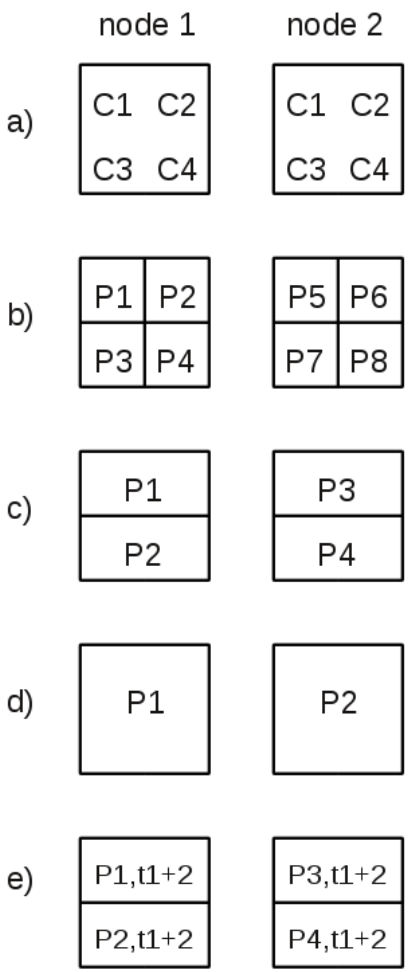
$$cost = O(N_{lon} \times N_{lat} \times N_{lev} \times N_{ens} \times N_{obs})$$

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.

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



Example computer cluster with 2 nodes, each having 4 cores



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
```

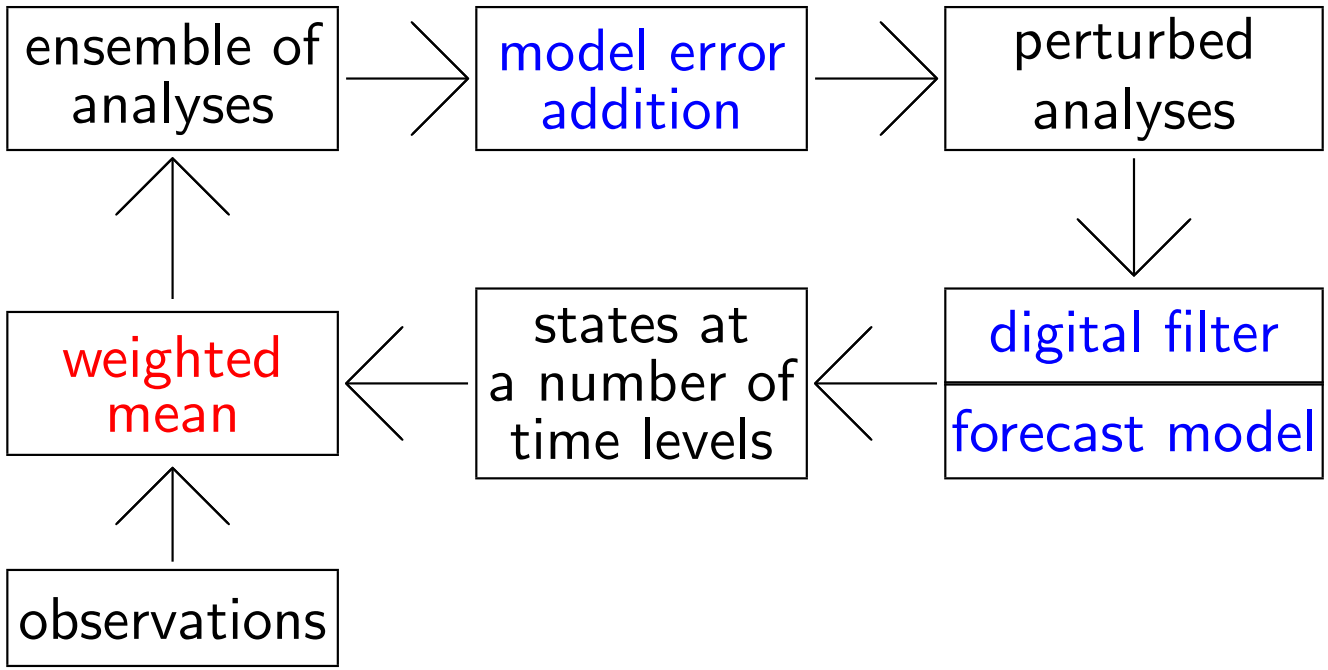
trends

The trend is towards having more compute cores with each computer upgrade. Future machines may well have O(100 000) cores available for EnKF applications.

The standard programming model is to use Fortran code with parallelization by means of a hybrid use of MPI and OpenMP.

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.

EnKF flowchart



For each of $N_{ens} = 256$ 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.

cost in a data-assimilation cycle

A data-assimilation cycle consists of three main steps:

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

With current (Canadian) 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.** It is not discussed further.

The forward operator

The forward operator needs to applied to each of the observations for each of the members of the ensemble. The operation count is thus proportional to N_{obs} and N_{ens}:

$$\text{Cost}(H) = O(N_{obs} N_{ens})$$

Pre-computing H prior to the computation of the analysis increment is convenient since it is independent for different members of the ensemble. Software to compute H from a complete model trajectory is likely available at an operational center.
Pre-computation of H is possible (and used) in most EnKF algorithms (LETKF as well as sequential algorithms).

The analysis increments: sequential algorithm

The weighted mean of the background and the observations is obtained using the Kalman Gain matrix:

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

By virtue of the sequential algorithm, the cost is dominated by

$$\mathrm{Cost}(\mathrm{P}_e^f \mathrm{H}^T) = O(N_{obs} N_{ens} N_{model})$$

With $N_{model} >> N_{obs}$, this matrix equation can be parallelized by dividing the model grid over the different processus.

Variations on the sequential algorithm are in operational use at CMC and NCEP.

The analysis increments: LETKF method

The LETKF computes weighting coefficients, at each grid point, to transform the background ensemble into an analysis ensemble (Szunyogh et al. ,2008, Tellus).

The cost is dominated by a term $(\mathrm{H}x)\mathrm{R}^{-1}(\mathrm{H}x)^T$

The cost depends on the local number of observations. It can be reduced by interpolation of weights.

An early inter-comparison of the LETKF with a sequential algorithm (Whitaker and Szunyogh) showed

1. perfect scaling for the LETKF,
2. better than perfect scaling for the sequential algorithm,
3. lower computational cost for the sequential algorithm.

Different results could be obtained with different parameters, implementations or different computer systems.

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.

Amdahl's law

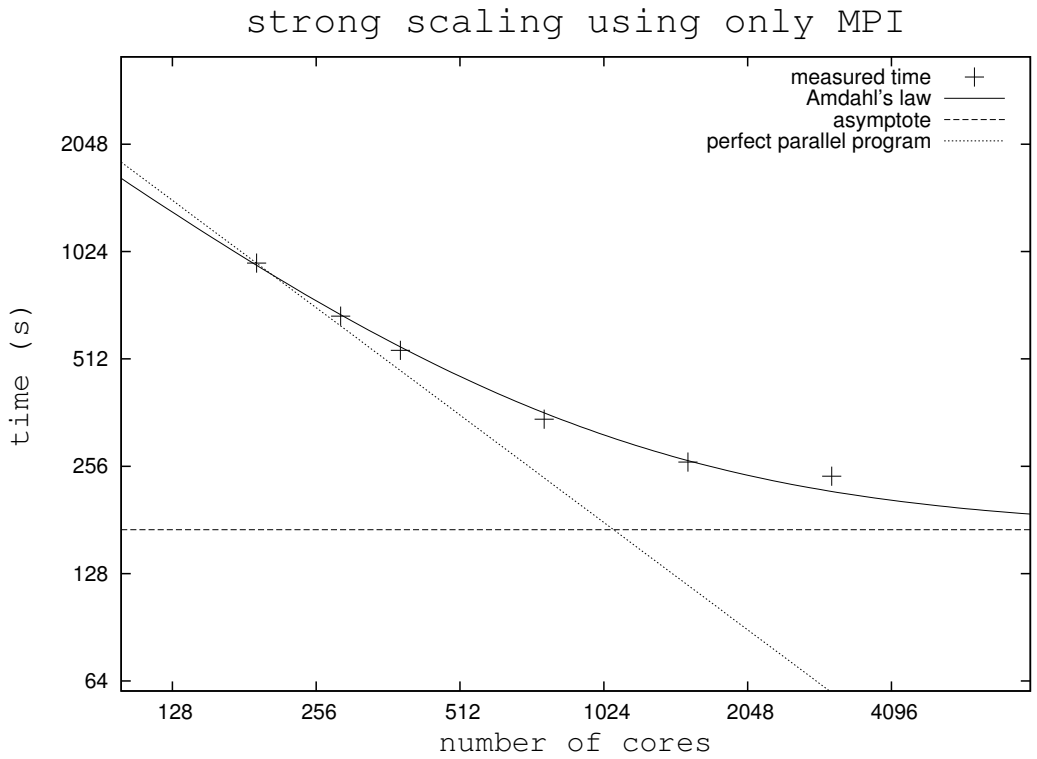
Given a sequential fraction s and N CPUs, the application speedup S is:

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

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

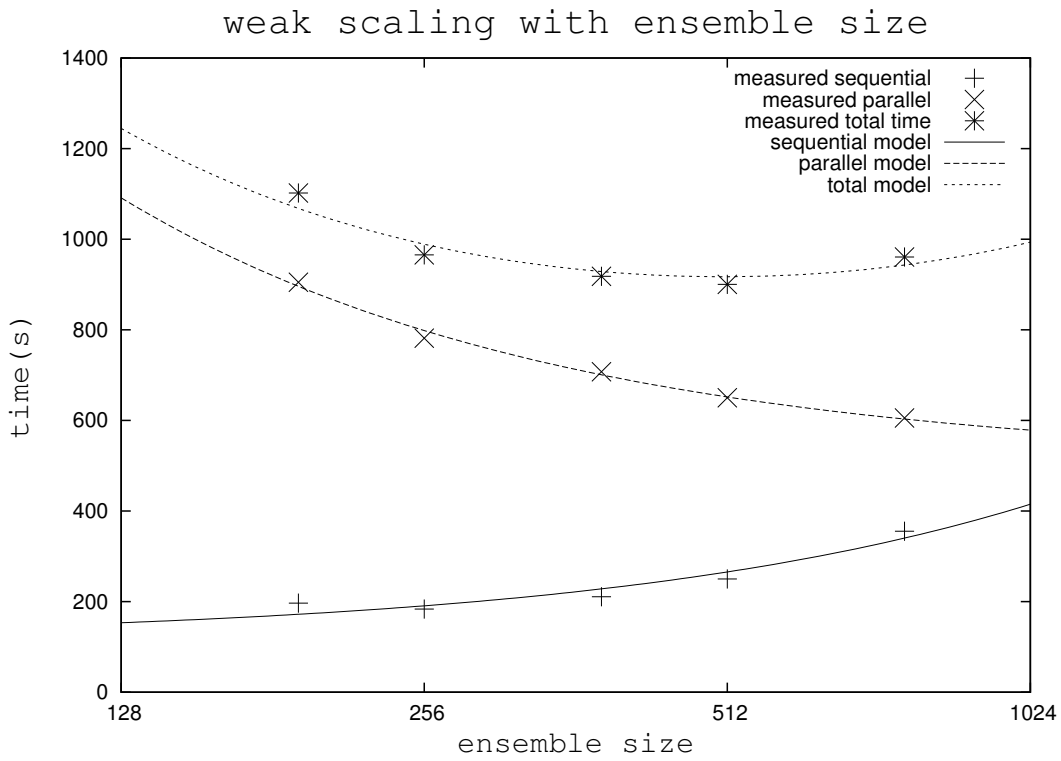
Good strong scaling is difficult because a very high fraction of the code needs to be parallelized. In practice, once we have reasonably fast execution, we are more likely to want to increase the problem size to obtain better results.

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.

software projects

1. the transition towards the use of shared Fortran-90 modules. We unified the handling of observational data between the En-Var and EnKF group. The unification of forward operators is in progress.
2. With the expected adoption of a higher resolution global EnKF, a global En-Var and a regional En-Var, we will no longer need to maintain the global and regional 4D-Var packages.
3. On our next supercomputer, we can likely use $O(10\,000)$ cores for the EnKF. Since this computer has an as yet unknown architecture it is an open question if our code can be made to run well on it.

For potential postdocs, it is very important to have experience with a programming language like Fortran-90.

summary and conclusion

1. The substantial computational cost of an EnKF makes it necessary to have an efficient implementation.
2. With more members, more observations or more grid points one will in general obtain higher quality results.
3. The model integration step is embarrashingly parallel.
4. Both sequential and local algorithms have been proposed.
5. To obtain an efficient analysis algorithm, it is necessary to measure the cost of the important components of the algorithm and to make adjustments where necessary. Optimization can have a very significant impact.
6. Computational cost depends on many aspects of the basic algorithm, the implementation and the computer system.
7. Future computers may have O(100 000) processors and having efficient parallel code will be very important.

Thank you