Data Assimilation and Scalability at ECMWF

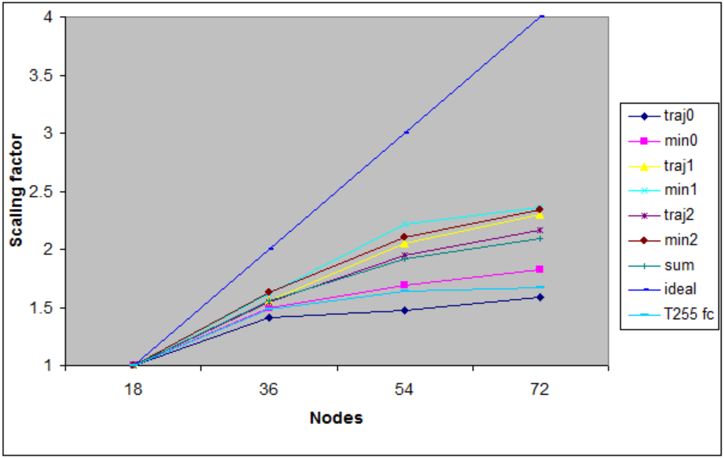
Yannick Trémolet

ECMWF

14 April 2014

With contributions from M. Fisher, S. Gürol, M. Hamrud, D. Salmond.

Scalability of 4D-Var

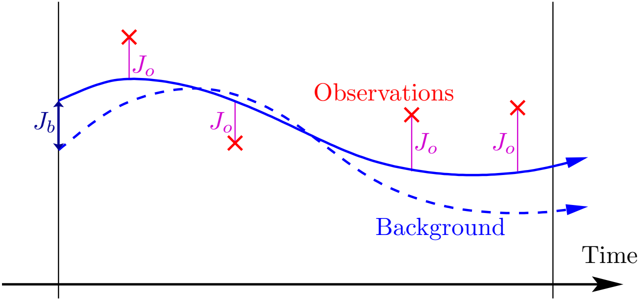


Data Assimilation on new and future architectures

- The overall cost of 4D-Var is approximately the same as one 10-day HRES forecast.
- We need to prepare our data assimilation system for future supercomputers: refactoring for *many core* architectures is unavoidable.
- The forecast model is one important component of the data assimilation system (80% of runtime): improvements in the model (and TL/AD) will benefit data assimilation directly.
- The same code adaptations as in the model will be used in other parts of the system (observation operators, covariance matrices, ...)
- However, the lower resolution of most of the data assimilation system ($\approx$20 times less grid-points) makes scalability even more problematic.

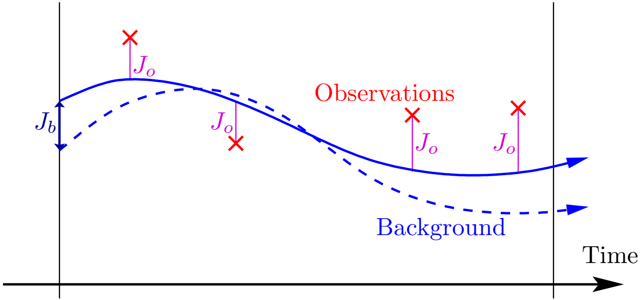
Parallelism in Data Assimilation

- On the positive side, some aspects of data assimilation that are not available in the forecast model can be exploited.



Parallelism in Data Assimilation

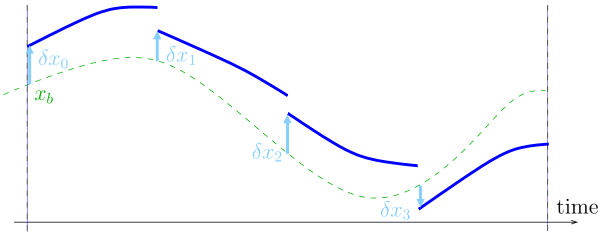
- On the positive side, some aspects of data assimilation that are not available in the forecast model can be exploited.



- The observations and background state are available throughout the whole window when we start the assimilation: it is possible to envisage parallelism in the time dimension.
- Assimilation should be considered a 4D problem, not an initial value problem.

Weak Constraint 4D-Var

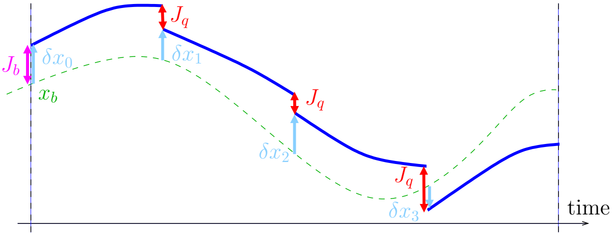
- The control variable is 4D, with some flexibility w.r.t. the resolution in time (it is already the case in the spatial dimensions).



- Model integrations within each time-step (or sub-window) are independent.
  - $\mathcal{M}$ and $\mathcal{H}$ can run in parallel for each time-step or sub-window.

Weak Constraint 4D-Var

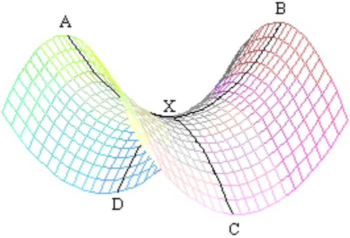
- The control variable is 4D, with some flexibility w.r.t. the resolution in time (it is already the case in the spatial dimensions).



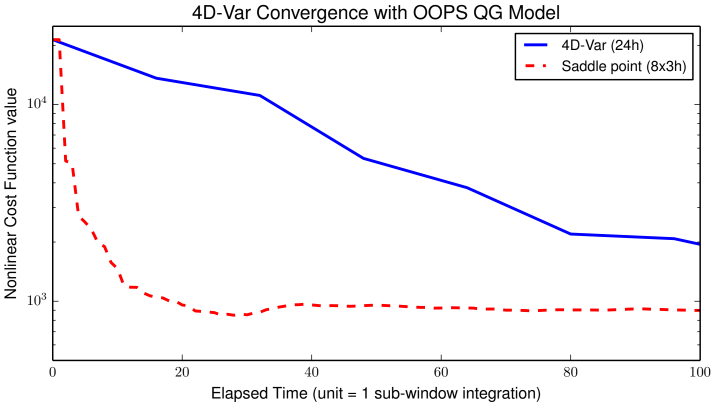
- Model integrations within each time-step (or sub-window) are independent.
  - $\mathcal{M}$ and $\mathcal{H}$ can run in parallel for each time-step or sub-window.
- The additional model error terms (J_q) make the minimization and preconditioning more complex: we need to explore dual (i.e. observation) or mixed primal/dual space algorithms.

Saddle-Point Formulation of 4D-Var

- The weak constraint 4D-Var problem can be written as a constrained minimisation problem.
- The inner loop minimization problem is replaced by a saddle point optimization problem (Lagrange multiplier approach).
- A few interesting properties are:
  - The inverses of the covariance matrices are not needed,
  - The parallelism over sub-windows is preserved,
  - The tangent linear and adjoint models can run in parallel.



Saddle-Point Formulation of 4D-Var



- The saddle point formulation of 4D-Var is more scalable.
- Weak constraint 4D-Var is also theoretically better then strong constraint 4D-Var although some questions remain (model error covariance matrix).

Ensemble Data Assimilation

- Ensemble methods obviously (and mostly) scale with the number of members.
- ECMWF uses an ensemble of 4D-Vars to estimate background error statistics.
- An alternative: EnKF
  - Approximations of a Kalman filter with covariances projected on ensemble space, with issues related to localisation in observation space.
  - A research implementation is maintained at ECMWF.
- An alternative: 4D-En-Var
  - Approximation of 4D-Var where time evolution of increments and covariances are projected on ensemble space (with localization),
  - Available in OOPS.
- ECMWF (like most operational centres) configuration is hybrid:
  - Choice of ensemble DA system for computing background error covariances: scalability is not the main concern.
  - 4D-Var provides the best high resolution analysis: we are improving its scalability.

Object-Oriented Programming

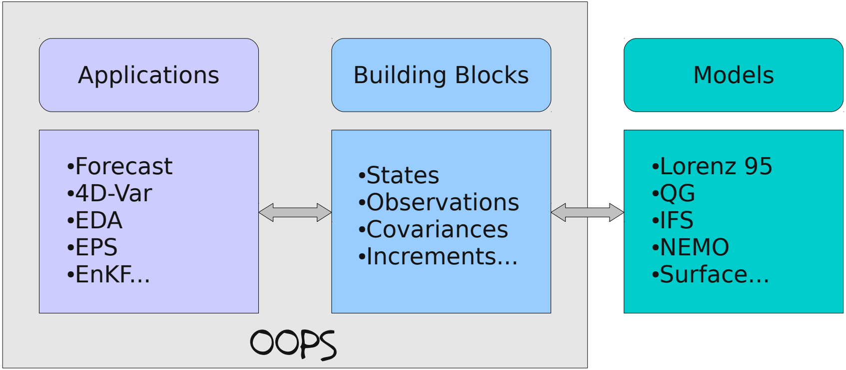
- Exploring parallelism in new directions, through weak constraint 4D-Var, new minimization algorithms or other techniques, requires considerable changes in the high level data assimilation algorithm.

- All that while getting ready for potential dramatic changes in the model...

- We need a very flexible, reliable, efficient, readable and modular code.
  - Readability improves staff efficiency: it is as important as computational efficiency (it's just more difficult to measure).
  - Modularity improves staff scalability: it is as important as computational scalability (it's just more difficult to measure).

- This is not specific to the IFS: the techniques that have emerged in the software industry to answer these needs are called **generic** and **object-oriented** programming.

Object-Oriented Prediction System



- The high levels Applications use abstract building blocks.
- The Models implement the building blocks.
- OOPS is independent of the Model being driven.

From IFS to OOPS

- The main idea is to keep the computational parts of the existing code and reuse them in a re-designed flexible structure.
- This can be achieved by a top-down and bottom-up approach.
  - From the top: Develop a new, modern, flexible structure (C++).
  - From the bottom: Progressively create self-contained units of code (Fortran).
  - Put the two together: Extract self-contained parts of the IFS and plug them into OOPS.
- From a Fortran point of view, this implies:
  - No global variables,
  - Control via interfaces (derived types passed by arguments).
- The OO layer developed for the simple models is not only a proof of concept: the same code is re-used to drive the IFS (generic).

OOPS Benefits

- Code components are independent:
  - Components can easily be developed in parallel.
  - Their complexity decreases: less bugs and easier testing and debuging.
- Improved flexibility:
  - Explore and improve scalability.
  - Develop new data assimilation (and other) science.
  - Changes in one application do not affect other applications.
  - Ability to handle different models opens the door for coupled DA.
- Simplified systems are very useful to understand concepts and validate ideas.
  - It is possible to move to the full system without re-writing code.
- Object-oriented programming does not solve scientific problems in itself: it provides a more powerful way to "tell the computer what to do".
  - For example, the saddle point formulation of weak constraint 4D-Var in OOPS works for any model.

Final comments

- Improved scalability will come from a dual approach:
  - Low level code adaptation to accelerator, GPU, many-cores architectures,
  - High level algorithmic and scientific changes to expose more parallelism.

- OOPS
  - Brings the necessary flexibility at the highest level.
  - Should a similar OO/template approach be used to encapsulate low level (possibly hardware dependent) optimized code?

- It is very likely that C++ will be used more and more:
  - OOPS control structure at the top level,
  - Data structures at low level,
  - "Scientific" code remaining in Fortran in between.