Parallelisation in the time dimension of 4D-Var

Mike Fisher[1]

ECMWF

1 December 2014

[1] Acknowledgement: Selime Gürol

Scalability of 4D-Var

The computational cost of 4D-Var is dominated by the cost of the linear and adjoint model integrations.

So, why does the forecast model scale well as the number of processors increases, but 4D-Var scales badly?

Characterisation of the forecast model

A typical forecast model for NWP:

- has a grid with $O(10^6)$ vertical columns
- has a timestep of $O(10^3)$ seconds
- produces forecasts $O(10^6)$ seconds ($\approx$10 days) ahead.
  - $\Rightarrow$ a forecast requires O(1000) timesteps

To be useful, the forecast must be produced within O(1) hour.

To achieve this, the model is parallelised over O(1000) processors.

Each processor performs calculations for O(1000) grid columns

Parallelising the forecast model

NWP models are currently parallelised in the horizontal only.

Each processor is assigned a number of grid columns, and performs the calculations for all the levels and all the timesteps of those columns.

But, increases in resolution are usually accompanied by:

- decreases in timestep (for numerical stability and/or accuracy)
- increases in the number of levels (to keep a reasonable ratio of vertical/horizontal resolution)

So, as resolution increases, there is more work per grid column.

To produce the forecast within the required O(1) hour, we must give each processor fewer grid columns to process (and employ more processors).

Parallelising the forecast model

If we assume that the number of vertical levels and the number of timesteps required to produce the forecast are both proportional to $\sqrt{\text{total number of grid columns}}$, then:

work per grid column $\propto$ total number of grid columns

grid columns per processor $\propto 1/(\text{total number of grid columns})$

Parallelising the forecast model

Current global NWP models assign O(1000) grid columns per processor.

Inter-processor communication and halo calculations start to dominate over computation if we have fewer than O(10) grid columns per processor.

This will happen once models reach resolutions O(10) finer than current models.

I.e. the current approach will start to fail when models reach O(1km) global resolution.

Characterisation of 4D-Var

Now consider a typical 4D-Var:

- Each inner-loop iteration involves two 12 hour integrations (TL and adjoint).
  - $\Rightarrow$ O(10^5) seconds of forecast per iteration.
- The timestep is O(10^3) seconds.
  - $\Rightarrow$ O(100) timesteps per iteration.
- An analysis requires O(100) iterations.
  - $\Rightarrow$ 4D-Var requires O(10^4) timesteps

To be useful, the analysis must be produced within O(1) hour.

4D-Var performs O(10) times more timesteps than the forecast model, but must run in a similar time.

Parallelising 4D-Var

To run 10 times more timesteps in the available time, we reduce the number of grid columns by a factor of 10 (by running at lower resolution).

4D-Var has O(100) grid columns per processor.

In a multi-incremental analysis, the low-resolution minimisations may have O(10) grid columns per processor.

(For example, the first minimisation of the ECMWF 4D-Var has ≈ 20 grid columns per processor.)

Parallelising 4D-Var

The problem of parallelising 4D-Var is not fundamentally different to that of parallelising the forecast model.

- 4D-Var benefits directly from improvements in the parallelisation of the forecast model.

In both cases, the current (horizontal-only) approach eventually fails because the number of processors required increases faster than the number of grid columns.

The forecast model can continue with the current approach for another 10–20 years.

The inner-loops of 4D-Var are already running out of parallelism.

Horizontal-only paralellisation is no longer enough. We need to find new dimensions to parallelise.

Weak-constraint 4D-Var

Let us define the analysis window as $t_0 \leq t \leq t_{N+1}$

We wish to estimate the sequence of states $x_0 \ldots x_N$ (valid at times $t_0 \ldots t_N$), given:

- A prior x_b (valid at t_0).
- A set of observations $y_0 \ldots y_N$.
  Each y_k is a vector containing, typically, a large number of measurements of a variety of variables distributed spatially and in the time interval $[t_k \ , \ t_{k+1})$.

4D-Var is a maximum likelihood method. We define the estimate as the sequence of states that minimizes the cost function:

$$J(x_0 \ldots x_N) = -\log\left(p(x_0 \ldots x_N | x_b; y_0 \ldots y_N)\right) + const.$$

Weak-constraint 4D-Var

Using Bayes' theorem, and assuming unbiased Gaussian errors, the weak-constraint 4D-Var cost function can be written as:

$$
\begin{aligned}
J(x_0 \dots x_N) \quad = \quad & \frac{1}{2}(x_0 - x_b)^{\mathrm{T}} B^{-1} (x_0 - x_b) \\
& + \frac{1}{2} \sum_{k=0}^{N} (\mathcal{H}_k(x_k) - y_k)^{\mathrm{T}} R_k^{-1} (\mathcal{H}_k(x_k) - y_k) \\
& + \frac{1}{2} \sum_{k=1}^{N} (q_k - \bar{q})^{\mathrm{T}} Q_k^{-1} (q_k - \bar{q}) .
\end{aligned}
$$

where $q_k = x_k - \mathcal{M}_k(x_{k-1})$

B, R_k and Q_k are covariance matrices of background, observation and model error. $\mathcal{H}_k$ is an operator that maps model variables x_k to observed variables y_k, and $\mathcal{M}_k$ represents an integration of the numerical model from time t_{k-1} to time t_k.

Weak Constraint 4D-Var: Quadratic Inner Loops

The inner loops of incremental weak-constraint 4D-Var minimise:

$$
\begin{aligned}
J(\delta x_0, \ldots, \delta x_N) \quad = \quad & \frac{1}{2}\left(\delta x_0 - b\right)^{\mathrm{T}} B^{-1} \left(\delta x_0 - b\right) \\
& + \frac{1}{2} \sum_{k=0}^{N} \left(H_k \delta x_k - d_k\right)^{\mathrm{T}} R_k^{-1} \left(H_k \delta x_k - d_k\right) \\
& + \frac{1}{2} \sum_{k=1}^{N} \left(\delta q_k - c_k\right)^{\mathrm{T}} Q_k^{-1} \left(\delta q_k - c_k\right)
\end{aligned}
$$

where $\delta q_k = \delta x_k - M_k \delta x_{k-1}$,
and where b, c_k and d_k come from the outer loop:

$$
\begin{aligned}
b &= x_b - x_0 \\
c_k &= \bar{q} - q_k \\
d_k &= y_k - \mathcal{H}_k(x_k)
\end{aligned}
$$

Weak Constraint 4D-Var: Quadratic Inner Loops

We simplify the notation by defining some 4D vectors and matrices:

$$\delta \mathbf{x} = \begin{pmatrix} \delta x_0 \\ \delta x_1 \\ \vdots \\ \delta x_N \end{pmatrix} \qquad \delta \mathbf{p} = \begin{pmatrix} \delta x_0 \\ \delta q_1 \\ \vdots \\ \delta q_N \end{pmatrix}$$

These vectors are related through $\delta q_k = \delta x_k - M_k \delta x_{k-1}$.

We can write this relationship in matrix form as:

$$\delta \mathbf{p} = \mathbf{L} \delta \mathbf{x}$$

where:

$$\mathbf{L} = \begin{pmatrix} I & & & & \\ -M_1 & I & & & \\ & -M_2 & I & & \\ & & \ddots & \ddots & \\ & & & -M_N & I \end{pmatrix}$$

Weak Constraint 4D-Var: Quadratic Inner Loops

We will also define:

$$\mathbf{R} = \begin{pmatrix} R_0 & & & \\ & R_1 & & \\ & & \ddots & \\ & & & R_N \end{pmatrix}, \qquad \mathbf{D} = \begin{pmatrix} B & & & \\ & Q_1 & & \\ & & \ddots & \\ & & & Q_N \end{pmatrix},$$

$$\mathbf{H} = \begin{pmatrix} H_0 & & & \\ & H_1 & & \\ & & \ddots & \\ & & & H_N \end{pmatrix}, \qquad \mathbf{b} = \begin{pmatrix} b \\ c_1 \\ \vdots \\ c_N \end{pmatrix} \qquad \mathbf{d} = \begin{pmatrix} d_0 \\ d_1 \\ \vdots \\ d_N \end{pmatrix}.$$

ECMWF

Weak Constraint 4D-Var: Quadratic Inner Loops

With these definitions, we can write the inner-loop cost function as

$$J = \frac{1}{2}(\delta\mathbf{p} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\delta\mathbf{p} - \mathbf{b}) + \frac{1}{2}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})$$

Eliminating $\delta\mathbf{p}$ using $\delta\mathbf{p} = \mathbf{L}\delta\mathbf{x}$ allows us to express J as a function of $\delta\mathbf{x}$:

$$\boxed{J(\delta\mathbf{x}) = \frac{1}{2}(\mathbf{L}\delta\mathbf{x} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{L}\delta\mathbf{x} - \mathbf{b}) + \frac{1}{2}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})}$$

Alternatively, we can express J as a function of $\delta\mathbf{p}$:

$$\boxed{J(\delta\mathbf{p}) = \frac{1}{2}(\delta\mathbf{p} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\delta\mathbf{p} - \mathbf{b}) + \frac{1}{2}(\mathbf{H}\mathbf{L}^{-1}\delta\mathbf{p} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\mathbf{L}^{-1}\delta\mathbf{p} - \mathbf{d})}$$

Weak Constraint 4D-Var: Quadratic Inner Loops

$$\mathbf{L} = \begin{pmatrix} I & & & & \\ -M_1 & I & & & \\ & -M_2 & I & & \\ & & \ddots & \ddots & \\ & & & -M_N & I \end{pmatrix}$$

$\delta\mathbf{p} = \mathbf{L}\delta\mathbf{x}$ can be done in parallel: $\delta q_k = \delta x_k - M_k \delta x_{k-1}$.

We know all the $\delta x_{k-1}{}'s$. We can apply all the $M_k{}'s$ simultaneously.

An algorithm involving only $\mathbf{L}$ is time-parallel.

Weak Constraint 4D-Var: Quadratic Inner Loops

$$\mathbf{L} = \begin{pmatrix} I & & & & \\ -M_1 & I & & & \\ & -M_2 & I & & \\ & & \ddots & \ddots & \\ & & & -M_N & I \end{pmatrix}$$

$\delta\mathbf{x} = \mathbf{L}^{-1}\delta\mathbf{p}$ is sequential: $\delta x_k = M_k \delta x_{k-1} + \delta q_k$.

We have to generate each δx_{k-1} in turn before we can apply the next M_k.

An algorithm involving $\mathbf{L}^{-1}$ is sequential.

Forcing Formulation

$$\boxed{J(\delta\mathbf{p}) = \frac{1}{2}(\delta\mathbf{p} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\delta\mathbf{p} - \mathbf{b}) + \frac{1}{2}(\mathbf{H}\mathbf{L}^{-1}\delta\mathbf{p} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\mathbf{L}^{-1}\delta\mathbf{p} - \mathbf{d})}$$

The form of cost function resembles that of strong-constraint 4D-Var, and it can be minimised using techniques that have been developed for strong-constraint 4D-Var.

In particular, we can precondition it using $\mathbf{D}^{1/2}$ to diagonalise the first term:

$$J(\chi) = \frac{1}{2}\chi^{\mathrm{T}}\chi + \frac{1}{2}(\mathbf{H}\mathbf{L}^{-1}\delta\mathbf{p} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\mathbf{L}^{-1}\delta\mathbf{p} - \mathbf{d})$$

where $\delta\mathbf{p} = \mathbf{D}^{1/2}\chi + \mathbf{b}$.

Unfortunately, this version of the cost function is sequential, since it contains $\mathbf{L}^{-1}$.

4D State Formulation

$$\boxed{J(\delta\mathbf{x}) = \frac{1}{2}(\mathbf{L}\delta\mathbf{x} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{L}\delta\mathbf{x} - \mathbf{b}) + \frac{1}{2}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})}$$

This version of the cost function is parallel. It does not contain $\mathbf{L}^{-1}$.

Unfortunately, it is difficult to precondition.

4D State Formulation

$$\boxed{J(\delta\mathbf{x}) = \frac{1}{2}(\mathbf{L}\delta\mathbf{x} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{L}\delta\mathbf{x} - \mathbf{b}) + \frac{1}{2}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})}$$

The usual method of preconditioning used in 4D-Var defines a control variable χ that diagonalizes the first term of the cost function

$$\delta\mathbf{x} = \mathbf{L}^{-1}(\mathbf{D}^{1/2}\chi + \mathbf{b})$$

With this change-of-variable, the cost function becomes:

$$J(\chi) = \frac{1}{2}\chi^{\mathrm{T}}\chi + \frac{1}{2}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})$$

But, we have introduced a sequential model integration (i.e. $\mathbf{L}^{-1}$) into the preconditioner.

4D State Formulation

Since $\mathbf{L}^{-1}$ appears only in the preconditioner, it is tempting to replace it by something cheaper.

Unfortunately, this destroys the preconditioning, due to the extreme ill-conditioning of $\mathbf{D}$. (See: Haben et al 2011)

If we approximate $\mathbf{L}$ by $\tilde{\mathbf{L}}$ in the preconditioner, the Hessian matrix of the first term of the cost function becomes

$$\mathbf{D}^{1/2}\tilde{\mathbf{L}}^{-\mathrm{T}}\mathbf{L}^{\mathrm{T}}\mathbf{D}^{-1}\mathbf{L}\tilde{\mathbf{L}}^{-1}\mathbf{D}^{1/2}$$

Because $\mathbf{D}$ is highly ill-conditioned, the inner $\mathbf{D}^{-1}$ does not cancel the outer $\mathbf{D}^{1/2}$'s, and the Hessian remains ill conditioned unless $\tilde{\mathbf{L}} = \mathbf{L}$.

Lagrangian Dual (4D-PSAS)

A third possibility for minimising the cost function is the Lagrangian dual (known as 4D-PSAS in the meteorological community):

$$\delta \mathbf{x} = \mathbf{L}^{-1}\mathbf{D}\mathbf{L}^{-\mathrm{T}}\mathbf{H}^{\mathrm{T}}\mu + \mathbf{L}^{-1}\mathbf{b}$$

where μ minimises:

$$\Phi(\mu) = \frac{1}{2}\mu^{\mathrm{T}}(\mathbf{R} + \mathbf{H}\mathbf{L}^{-1}\mathbf{D}\mathbf{L}^{-\mathrm{T}}\mathbf{H}^{\mathrm{T}})\mu + \mu(\mathbf{H}\mathbf{L}^{-1}\mathbf{b} - \mathbf{d})$$

Clearly, this is a sequential algorithm, since it contains $\mathbf{L}^{-1}$.

The Saddle Point Formulation

We have seen that, of the standard formulations of 4D-Var, only the 4D-state formulation is capable of being parallelised in the time dimension.

However, the 4D-state formulation is difficult to precondition.

The saddle point formulation is a new formulation of 4D-Var that is both time-parallel and can be preconditioned efficiently.

The Saddle Point Formulation

$$\boxed{J(\delta\mathbf{x}) = \frac{1}{2}(\mathbf{L}\delta\mathbf{x} - \mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{L}\delta\mathbf{x} - \mathbf{b}) + \frac{1}{2}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\delta\mathbf{x} - \mathbf{d})}$$

At the minimum:

$$\nabla J = \mathbf{L}^{\mathrm{T}}\mathbf{D}^{-1}(\mathbf{L}\delta\mathbf{x} - \mathbf{b}) + \mathbf{H}^{\mathrm{T}}\mathbf{R}^{-1}(\mathbf{H}\delta\mathbf{x} - \mathbf{d}) = \mathbf{0}$$

Define:

$$\lambda = \mathbf{D}^{-1}(\mathbf{b} - \mathbf{L}\delta\mathbf{x}), \qquad \mu = \mathbf{R}^{-1}(\mathbf{d} - \mathbf{H}\delta\mathbf{x})$$

Then:

$$\left.\begin{array}{rcl} \mathbf{D}\lambda + \mathbf{L}\delta\mathbf{x} & = & \mathbf{b} \\ \mathbf{R}\mu + \mathbf{H}\delta\mathbf{x} & = & \mathbf{d} \\ \mathbf{L}^{\mathrm{T}}\lambda + \mathbf{H}^{\mathrm{T}}\mu & = & \mathbf{0} \end{array}\right\} \Longrightarrow \begin{pmatrix} \mathbf{D} & \mathbf{0} & \mathbf{L} \\ \mathbf{0} & \mathbf{R} & \mathbf{H} \\ \mathbf{L}^{\mathrm{T}} & \mathbf{H}^{\mathrm{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \\ \delta\mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{d} \\ \mathbf{0} \end{pmatrix}$$

Saddle Point Formulation

$$\boxed{\begin{pmatrix} \mathbf{D} & \mathbf{0} & \mathbf{L} \\ \mathbf{0} & \mathbf{R} & \mathbf{H} \\ \mathbf{L}^{\mathrm{T}} & \mathbf{H}^{\mathrm{T}} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \lambda \\ \mu \\ \delta\mathbf{x} \end{pmatrix} = \begin{pmatrix} \mathbf{b} \\ \mathbf{d} \\ \mathbf{0} \end{pmatrix}}$$

We call this the saddle point formulation of weak-constraint 4D-Var.

The block 3×3 matrix is a saddle point matrix. It is real, symmetric, indefinite.

Note that the matrix contains no inverse matrices.

- We can apply the matrix without requiring multiplication by $\mathbf{L}^{-1}$.

The saddle point formulation is time paralel.

Saddle Point Formulation

Another way to derive the saddle point formulation is to regard the minimisation as a constrained problem:

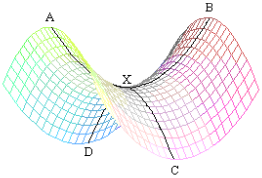
$$\begin{aligned}\min_{\delta\mathbf{p},\delta\mathbf{w}} J(\delta\mathbf{p},\delta\mathbf{w}) &= (\delta\mathbf{p}-\mathbf{b})^{\mathrm{T}}\mathbf{D}^{-1}(\delta\mathbf{p}-\mathbf{b}) + (\delta\mathbf{w}-\mathbf{d})^{\mathrm{T}}\mathbf{R}^{-1}(\delta\mathbf{w}-\mathbf{d}) \\ &\quad \text{subject to } \delta\mathbf{p} = \mathbf{L}\delta\mathbf{x} \text{ and } \delta\mathbf{w} = \mathbf{H}\delta\mathbf{x}.\end{aligned}$$

Introducing Lagrange multipliers λ and μ for the constraints gives the Lagrangian:

$$\mathcal{L}(\delta\mathbf{x},\delta\mathbf{p},\delta\mathbf{w},\lambda,\mu) = J + \lambda^{\mathrm{T}}(\delta\mathbf{p}-\mathbf{L}\delta\mathbf{x}) + \mu^{\mathrm{T}}(\delta\mathbf{w}-\mathbf{H}\delta\mathbf{x})$$

Setting the gradient of this function to zero gives a system of 5 linear equations, which we can reduce to 3 by eliminating $\delta\mathbf{p}$ and $\delta\mathbf{w}$.

Saddle Point Formulation



Lagrangian: $\mathcal{L}(\delta\mathbf{x}, \delta\mathbf{p}, \delta\mathbf{w}, \lambda, \mu)$

4D-Var solves the primal problem: minimise along AXB.

4D-PSAS solves the Lagrangian dual problem: maximise along CXD.

The saddle point formulation finds the saddle point of $\mathcal{L}$.

The saddle point formulation is neither 4D-Var nor 4D-PSAS.

Saddle Point Formulation

To solve the saddle point system, we have to precondition it.

Preconditioning saddle point systems is the subject of much current research.

- See e.g. Benzi and Wathen (2008), Benzi, Golub and Liesen (2005).

One possibility (c.f. Bergamaschi, *et al.*, 2011) is to approximate the saddle point matrix by:

$$\tilde{\mathcal{P}} = \begin{pmatrix} \mathbf{D} & \mathbf{0} & \tilde{\mathbf{L}} \\ \mathbf{0} & \mathbf{R} & \mathbf{0} \\ \tilde{\mathbf{L}}^{\mathrm{T}} & \mathbf{0} & \mathbf{0} \end{pmatrix} \quad \Rightarrow \quad \tilde{\mathcal{P}}^{-1} = \begin{pmatrix} \mathbf{0} & \mathbf{0} & \tilde{\mathbf{L}}^{-\mathrm{T}} \\ \mathbf{0} & \mathbf{R}^{-1} & \mathbf{0} \\ \tilde{\mathbf{L}}^{-1} & \mathbf{0} & -\tilde{\mathbf{L}}^{-1}\mathbf{D}\tilde{\mathbf{L}}^{-\mathrm{T}} \end{pmatrix}$$

Note that $\mathbf{D}^{-1}$ is not required.

Saddle Point Formulation

The experimental results shown in this talk used either $\tilde{\mathbf{L}} = \mathbf{L}$, or:

$$\tilde{\mathbf{L}} = \begin{pmatrix} I & & & & \\ -I & I & & & \\ & -I & I & & \\ & & \ddots & \ddots & \\ & & & -I & I \end{pmatrix}$$

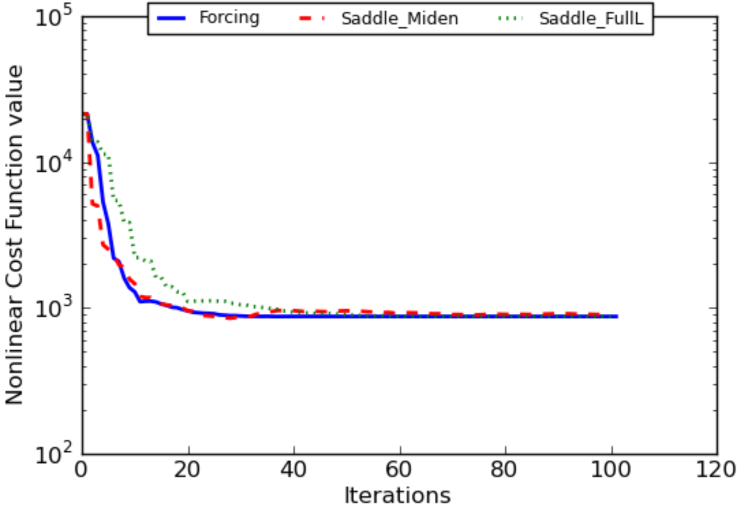
Selime will say much more about preconditioning the saddle point system in her talk.

Results from a toy system

- The practical results shown in the next few slides are for a simplified (toy) analogue of a real system.
- The model is a two-level quasi-geostrophic channel model with 1600 gridpoints.
- The model has realistic error-growth and time-to-nonlinearity
- There are 100 observations of streamfunction every 3 hours, plus 100 wind observations and 100 wind-speed observations every 4 hours.
- The error covariances are assumed to be horizontally isotropic and homogeneous, with a Gaussian spatial structure.
- The analysis window is 24 hours, and is divided into eight 3h subwindows.
- The solution algorithm was GMRES (implemented by Selime Gürol).
- Selime also ran the experiments.
- We used the Object-Oriented Prediction System (OOPS).

Saddle Point Formulation

Convergence as a function of iteration



Saddle Point Formulation

Even without parallelisation, the saddle point formulation is competitive with the forcing formulation.

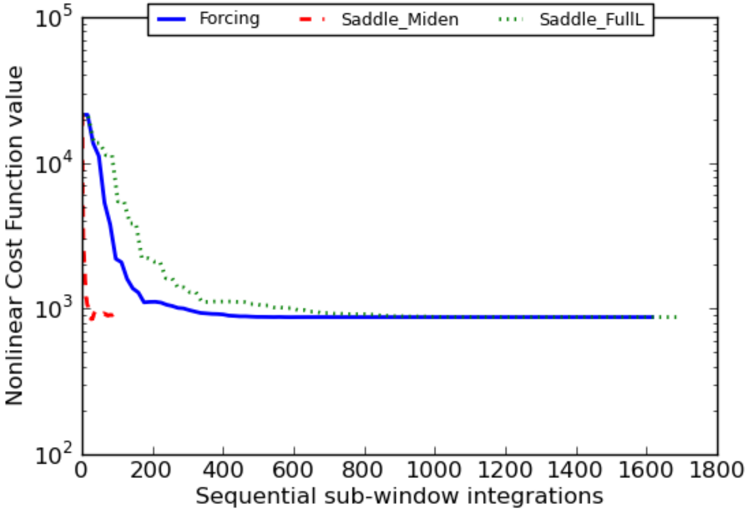
We can estimate the potential parallel speed-up by counting the number of sequential sub-window integrations required by the different formulations.

At each iteration, the forcing formulation performs 8 sequential sub-window integrations in the TL, followed by 8 in the adjoint.

The saddle point algorithm can run all 16 integrations in parallel.

Saddle Point Formulation

Convergence as a function of subwindow integrations ($\approx$ wallclock time)



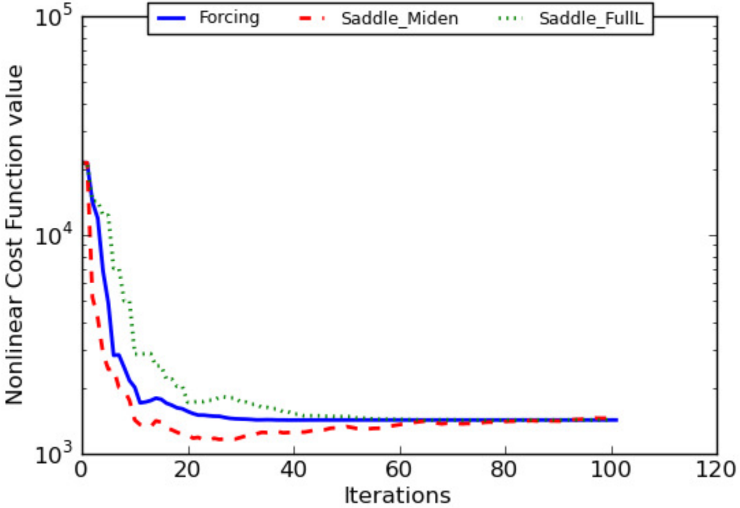
Conclusions

- Eventually, a horizontal-only approach to parallelising NWP models must fail.
- For the forecast model, this will happen when resolutions approach 1km (global).
- For 4D-Var, we are already there.
- The future viability of 4D-Var as an algorithm for Numerical Weather Prediction depends on finding, and exploiting, new dimensions of parallelism.
- The saddle point formulation of weak-constraint 4D-Var allows parallelisation in the time dimension.
- The algorithm is competitive with existing algorithms and has the potential to allow 4D-Var to remain computationally viable on next-generation computer architectures.

Backup Slides. . .

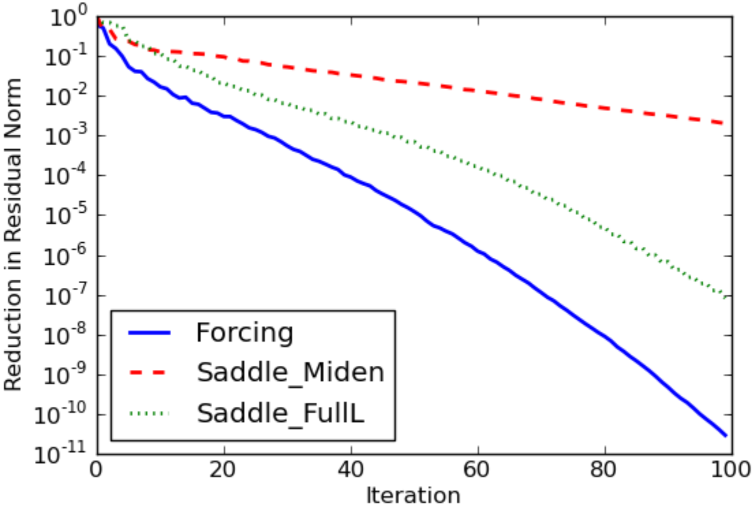
Saddle Point Formulation

Convergence as a function of iteration — 12 sub-windows



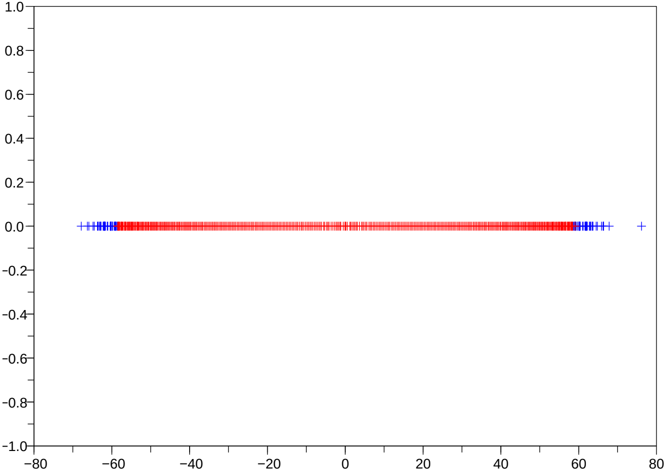
Saddle Point Formulation

Convergence of residual norms — 8 sub-windows



Saddle Point Formulation

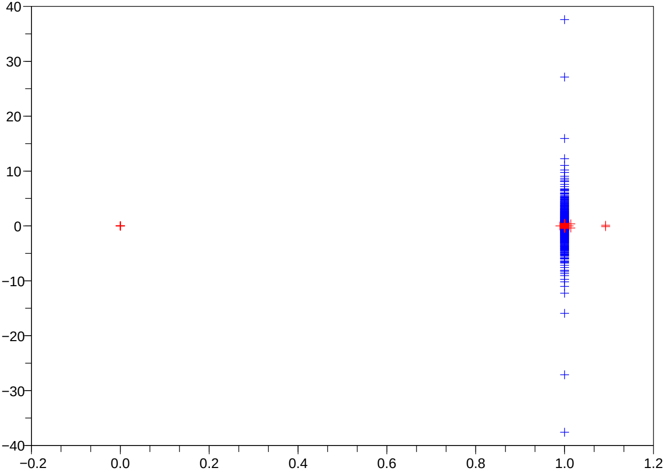
OOPS QG model. 24-hour window with 8 subwindows.



Ritz Values of $\mathcal{A}$.

Saddle Point Formulation

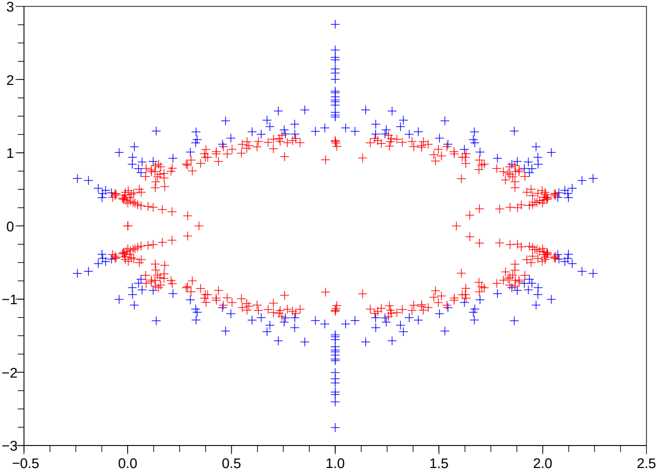
OOPS QG model. 24-hour window with 8 subwindows.



Ritz Values of $\tilde{\mathcal{P}}^{-1}\mathcal{A}$ for $\tilde{\mathbf{L}} = \mathbf{L}$.

Saddle Point Formulation

OOPS QG model. 24-hour window with 8 subwindows.



Ritz Values of $\tilde{\mathcal{P}}^{-1}\mathcal{A}$ for $\tilde{\mathbf{L}} = \mathbf{I}$.