# Strategies for parallel variational data assimilation

by

Jarmo Rantakokko

Report No. 181/1996

UPPSALA UNIVERSITET
Institutionen för
teknisk databehandling

UPPSALA UNIVERSITY
Department of
Scientific Computing

# Strategies for parallel variational data assimilation *

Jarmo Rantakokko

May 17, 1996

**Abstract**

The prospects to parallelize a variational data assimilation scheme from the starting point of an existing parallel forecast model and its adjoint equations have been investigated. Three parallelization strategies to do the implementation and two partitioning algorithms to compute data distributions have been suggested. Numerical simulations of the parallelizations show that the strategies and the partitioning algorithms can be combined to perform well on shared memory respectively distributed memory machines and that they give better results than direct use of standard parallelization methods.

*The research presented in this paper was supported by the Parallel and Scientific Computing Institute (PSCI).

# Contents

# 1 Introduction

Numerical weather prediction is very sensitive to disturbances in initial data. In the literature there are numerous examples where serious storms were poorly forecast because of small errors in initial data originating from erroneous or insufficient observations. Clearly, an accurate analysis of the state of the atmosphere is indispensable for forecasting. Lots of research is going on in this area and a new trend is to move into variational data assimilation techniques. These methods lead to enormous optimization problems with an order of $10^7$ variables. The estimated work of a four-dimensional (4D) variational data assimilation is about 50-100 times the work of a forecast run. With the iterative incremental approach suggested in [1], the work can probably be brought down to be in the range of 5-25 times the work for the forecast run up to 48 hours. Still, this is very expensive and in an operational use the computation time is strongly limited. With today's parallel computers this is not a prohibitive difficulty. Parallelization of these algorithms must then be considered from the beginning. The difficulty here is that the observations are sparsely and irregularly distributed in space and time. In this paper we will suggest and analyze some parallelization strategies for a 4D variational data assimilation scheme. The principles of the assimilation scheme are described in Section 2. In the third section different parallelization strategies are analyzed and in the following section we have suggested two new data partitioning algorithms, the Recursive Bisection method with Iterative Improvement and the Possessive Greedy method. Simulations of the different parallelizations are made in Section 5.

# 2 Four-dimensional variational data assimilation

The meteorological initial value problem is an undetermined system of equations. The variables in the flow model are for example the temperature, the surface pressure, the horizontal wind components, the specific humidity, and the cloud water content. In addition, a number of surface and soil variables, for example surface temperature, soil moisture, and snow depth, are involved. Depending on the resolution of the numerical model this gives $10^6$ - $10^7$ degrees of freedom. The observation data come from surface stations, radiosondes, radar stations, aircraft reports, and satellite soundings. The degree of freedom is on the order of $10^5$. Thus, we have a large undetermined system of equations and additional information of the flow between the observation points, a background field, is needed. Such information can be provided by a previous forecast run. Moreover, the initial value problem is complicated by errors in the observations and approximations in the models. Consequently, advanced statistical methods must be used to produce the initial state for the flow model.

Meteorological data assimilation is a process in which observations are merged together with a dynamical flow model in order to determine as accurate as possible a description of the state of the atmosphere. The governing equations of the flow follows essentially the principles of conservation of mass, energy, and momentum and are non-linear. The observations can be quantities used directly in the flow equations or they can be quantities that are functions of the equation quantities. Thus, the assimilation process is a large non-linear optimization problem.

The latest step in the progress of data assimilation techniques is the development of 4D variational data assimilation, where time is the fourth dimension. The algorithm computes an initial state of the atmosphere at a time $t_0$ that gives a model run over a time interval $t_0 \to t_1$ which is adjusted in an optimal way to all the observations in this time interval. A forecast run is then initiated from this optimized model run. The idea is to minimize a cost function,

$$J = J_B + J_O \tag{1}$$

where

$$J_B = \frac{1}{2}(x(t_0) - x_b)^T B^{-1} (x(t_0) - x_b) \tag{2}$$

is the background cost function and

$$J_O = \frac{1}{2}\sum_i (H_i x(t_i) - y_i)^T O^{-1} (H_i x(t_i) - y_i) \tag{3}$$

is the observation cost function. Here $x_b$ is the background field (e.g. a best possible short range forecast), $y_i$ the observations, $H_i$ a transformation and interpolation operator, $B$ and $O$ the background error covariance matrix and the observation error covariance matrix, respectively, and $x(t_0)$ the initial state. The cost function is minimized with respect to $x(t_0)$ under the constraint,

$$x(t_i) = M(t_0, t_i)\, x(t_0) \tag{4}$$

where $M(t_0, t_i)$ is the numerical flow model.

There are basically two methods for solving a constrained minimization problem, see for example [4]. One way is to transform the problem to an unconstrained problem by eliminating the constrained variable, as in [1] and [13]. The other method consists of using Lagrange multipliers, as in [2]. We will here only consider the method of eliminating the constrained variable. Thus, we insert (4) into (3) and get the unconstrained problem of minimizing (1) with respect to $x(t_0)$ where $J_O$ becomes

$$J_O = \frac{1}{2}\sum_i (H_i M(t_0, t_i)\, x(t_0) - y_i)^T O^{-1} (H_i M(t_0, t_i)\, x(t_0) - y_i)$$

Most of the minimization schemes imply computations of the gradient $\nabla_{x(t_0)} J$. It is impossible to establish an explicit analytical expression of the gradient to the numerical flow model, but with the method of adjoint equations it is feasible to approximate the gradient at a reasonable numerical cost. The idea can be described briefly as follows.

The model state at $t_0$ can be written as a sum of the background field and a small perturbation $\Delta x$

$$x(t_0) = x_b + \Delta x$$

The numerical flow model can then be linearized around $x_b$

$$x(t_i) = M(t_0, t_i)\, x(t_0) = M(t_0, t_i)(x_b + \Delta x) \approx M(t_0, t_i)\, x_b + T_i \Delta x$$

where $T_i = A_{n_i} \cdots A_2 A_1$ is a product of jacobians $A_k$ of $M$ in different time-steps $k$. The background cost function can now be written as

$$J_B = \frac{1}{2} \Delta x^T B^{-1} \Delta x \tag{5}$$

and the observation cost function as

$$J_O = \frac{1}{2} \sum_i \left( H_i \left( M x_b + T_i \Delta x \right) - y_i \right)^T O^{-1} \left( H_i \left( M x_b + T_i \Delta x \right) - y_i \right) \tag{6}$$

The gradient of J with respect to $\Delta x$ becomes

$$\nabla_{\Delta x} J = B^{-1} \Delta x + \sum_i T_i^T H_i^T O^{-1} \left( H_i \left( M x_b + T_i \Delta x \right) - y_i \right) \tag{7}$$

Here, $T_i$ is called the tangent linear model of $M$ and $T_i{}^T$ represents the adjoint equations. The adjoint equations can be seen as a backward integration of the gradient in time $t_i$ to time $t_0$, $\nabla_{x(t_0)} J \approx T_i^T \nabla_{x(t_i)} J = A_1^T A_2^T \cdots A_{n_i}^T \nabla_{x(t_i)} J$.

The numerical cost of the minimization is still too severe and further approximations are required. In [1] an iterative incremental formulation of the variational assimilation scheme is suggested. The method produces a sequence of model states

$$x^n \left( t_0 \right) = x_b^n + \Delta x$$

with

$$x_b^n = x_b + \Delta x_a^{n-1}$$

where $\Delta x$ is the result of the minimization in iteration $n$ and

$$\Delta x_a^n = \Delta x_a^{n-1} + \Delta x, \quad \Delta x_a^0 = 0$$

which implies that

$$x_b^{n+1} = x_b + \Delta x_a^n = x_b + \Delta x_a^{n-1} + \Delta x = x_b^n + \Delta x$$

is a refined background field. The numerical flow model is linearized as before

$$x^n \left( t_i \right) = M \left( x^n \left( t_0 \right) \right) \approx M \left( x_b^n \right) + T_i \Delta x$$

and the resulting cost functions becomes

$$J_B = \frac{1}{2} \left( x^n \left( t_0 \right) - x_b \right)^T B^{-1} \left( x^n \left( t_0 \right) - x_b \right) =$$

$$\frac{1}{2} \left( \Delta x_a^{n-1} + \Delta x \right)^T B^{-1} \left( \Delta x_a^{n-1} + \Delta x \right)$$

and

$$J_O = \frac{1}{2} \sum_i \left( H_i \left( M \left( x_b^n \right) + T_i \Delta x \right) - y_i \right)^T O^{-1} \left( H_i \left( M \left( x_b^n \right) + T_i \Delta x \right) - y_i \right)$$

The basic idea of rewriting the equations is to have different resolutions in the complete flow model and in the tangent linear and adjoint equations. Now, if we define $R$ (and its inverse $P$) as a restriction operator from the fine resolution grid to the coarse resolution grid introducing $z = R\Delta x$ as the new control variable and $G_i$ as the tangent linear operator on the coarse grid, then we can write the cost function as

$$J = \frac{1}{2}\left(z_a^{n-1} + z\right)^T B^{-1} \left(z_a^{n-1} + z\right) +$$

$$\frac{1}{2}\sum_i \left(H_i\left(M\left(x_b^n\right) + PG_i z\right) - y_i\right)^T O^{-1} \left(H_i\left(M\left(x_b^n\right) + PG_i z\right) - y_i\right) \qquad (8)$$

and the update of the refined background field

$$x_b^{n+1} = x_b^n + Pz \qquad (9)$$

The iteration scheme is concluded with

$$z_a^n = z_a^{n-1} + z, \quad z_a^0 = 0 \qquad (10)$$

Thus, the minimization is performed in a pair of nested loops. The outer loop uses the high resolution non-linear model to compute a refined model trajectory from time $t_0$ to $t_i$. The inner loop uses the tangent linear model and the adjoint equations of a lower resolution to minimize the cost function and to compute a refined model state at time $t_0$.

The computation of $B^{-1}$ can be reduced to simple transformations, first by assuming vertical-horizontal separability and then by assuming that the horizontal background error correlation is a function of distance only. Thus, by e.g. projection on eigenvectors of the vertical background error correlation matrix and by horizontal Fourier transforms, we can make a transformation $\chi = Q\Delta x$ so that $QBQ^*$ is diagonal and easy to invert, [9].

# 3 Parallelization strategies

## 3.1 Parallelization aspects

The task to be addressed here is to construct and analyze an algorithm for data partitioning in the cost function routine. The main work is in computing the observation error cost function and its gradient. The computation of the background error cost function and its gradient becomes trivial with the assumed transformations.

In our parallelization strategy we must consider the variable transformations, $Q^*$ and $(Q^*)^{-1}$, the non-linear model integration, $M_i$, the tangent-linear model integration, $T_i$, the adjoint equations, $T_i^T$, and the transformation from model variables to observed quantities, $H_i$ and $H_i^T$. The model integration routines, $M_i$, $T_i$, and $T_i^T$, are already parallelized subroutines with a fixed distribution of data. Also, the variable transformation, $Q^*$ and $(Q^*)^{-1}$, uses the same data distribution as the model routines. This is an additional constraint in the parallelization process. We must provide these routines with the specific data distribution. In an iterative incremental formulation of the assimilation scheme, equations (8) to (10), the full flow model and the tangent linear equations have different resolution and may need different data distributions.

The transformation procedure from model variables to observed quantities can be divided into two parts: one part where we interpolate the model variables to the observation points and one where the model variables are integrated to the same quantities as the observation variables. The second part can be done locally with vertical dependencies only. If we use the same data distribution in this transformation procedure as in the flow model the load balance will not be good. The number of observations, $y_i$, varies in space and time. In addition, the work in the variable transformation part can vary considerably for different types of observations.

The minimization is supposed to be performed with a quasi-Newton type method. This routine may also need its own data distribution. At the moment this routine is not parallelized; thus the data need to be "undistributed". The convergence of the minimization can be increased with a multi-grid approach. An approximate solution could first be computed on a course grid. This solution could then be prolongated to a finer grid and used as an initial condition or a background field. We then may have to have multiple data partitions, one for each grid.

In the system a quality control of the observations will also be integrated. This control could be included in the construction of the observation data files, or it could be a part of the assimilation procedure by using appropriate non-Gaussian observation error statistics. A few iterations with the minimization algorithm could also be performed in order to reject unfitted observations. Rejecting observations affects the load balance and we may have to redistribute data.

Finally, we must not focus on any particular computer architecture. The parallelization strategy should be general enough to work efficiently on different types of parallel computers. Thus, before implementing any parallel code it seems to be worth to investigate and compare different parallelization strategies theoretically.

### 3.2 Data distribution strategies

We will here focus on the basic assimilation scheme without any extensions to multi-grid techniques etc. There are then at least three conceivable strategies for a data parallel approach. The first alternative is to distribute the observations according to the distribution in the model integration routines. This data distribution will below be called the *model distribution*. In the second alternative, all data are distributed geographically in connected domains, balancing the work load in the procedure for transformation to observed quantities. This will be called the *geographic distribution*. Finally, in the third alternative two distributions are handled at the same time. One distribution for the variables at the grid-points, using the *model distribution*, and one for the observations, using a *free distribution*. A copy of those grid-point variables surrounding the observations is then distributed according to the distribution of the observations. Consequently, the observations can be more freely distributed (called the *free distribution*) than in connected domains.

We will describe the strategies in detail by breaking down the equations (5) to (7) into separate operations and commenting these operation steps.

**Strategy 1.**

| Operation | Comment |
| --- | --- |
| Input: $x_b$, $y$ | Background field and observations |
| Distribute $x_b$, $y$ | According to the model distribution |
| $\chi = 0$ | First guess |
| Do until convergence | Find minimum |
| Distribute $\chi$ | According to the model distribution |
| $J = \chi^* \chi$, $\nabla_\chi J = \chi$ | Compute $J_B$ and its gradient, local operations |
| $\Delta x = Q^{-1} \chi$ | Transform back, parallel routine using the model distribution |
| $x = M x_b + T \Delta x$ | Integrate non-linear model, parallel routines using the model distribution |
| For i=$n_t$,1,-1 | Loop to compute $J_O$ |
| Get $x_i$ | Get model state at time step $i$ |
| Update overlap in $x_i$ | Needed if $M$ based on spectral methods |
| $w_i = H_i x_i - y_i$ | Compute difference of the model state and the observations, local operation if partitions overlap |
| $J = J + w_i^* O^{-1} w_i$ | Add contribution, local operation if $O$ diagonal |
| $q_i = q_i + H_i^* O^{-1} w_i$ | Compute gradient at time step $i$, local operation if $O$ diagonal and partitions overlap |
| $q_{i-1} = T_{i,i-1}^* q_i$ | Integrate adjoint model from time i to i-1, parallel routine using the model distribution |
| End For | |
| $s_0 = (Q^{-1})^* q_0$ | Transform gradient, parallel routine using the model distribution |
| $\nabla_\chi J = \nabla_\chi J + s_0$ | Add $J_B$ and $J_O$, local sum |
| Undistribute $\nabla_\chi J$ and $J$ | Collect data |
| Call minimization routine | Calculate new $\chi$, serial routine |
| End Do | |

The advantage of the Strategy 1 is that it is easy to distribute data and that no re-distributions, other than arrangements of overlaps, are necessary. The drawback is the lack of load balance.

**Strategy 2.**

| Operation | Comment |
| --- | --- |
| Input: $x_b$, $y$ | As in Strategy 1 |
| Distribute $x_b$, $y$ | $y$ according to the geographic distribution |
| $\chi = 0$ | As in Strategy 1 |
| Do until convergence | As in Strategy 1 |
| Distribute $\chi$ | As in Strategy 1 |
| $J = \chi^* \chi$, $\nabla_\chi J = \chi$ | As in Strategy 1 |
| $\Delta x = Q^{-1} \chi$ | As in Strategy 1 |
| $x = M x_b + T \Delta x$ | As in Strategy 1 |
| For i=$n_t$,1,-1 | As in Strategy 1 |
| Get $x_i$ | As in Strategy 1 |

| | |
|---|---|
| Re-distribute $x_i$ | According to the geographic distribution |
| $w_i = H_i x_i - y_i$ | Local operation |
| $J = J + w_i^* O^{-1} w_i$ | Local operation if $O$ diagonal |
| $q_i = q_i + H_i^* O^{-1} w_i$ | Local operation if $O$ diagonal |
| Re-distribute $q_i$ | According to the model distribution |
| $q_{i-1} = T_{i,i-1}^* q_i$ | As in Strategy 1 |
| End For | |
| $s_0 = (Q^{-1})^* q_0$ | As in Strategy 1 |
| $\nabla_\chi J = \nabla_\chi J + s_0$ | As in Strategy 1 |
| Undistribute $\nabla_\chi J$ and $J$ | As in Strategy 1 |
| Call minimization routine | As in Strategy 1 |
| End Do | |

Now we have a good load balance but must do costly re-distributions of the grid-point variables. Moreover, the data partitioning algorithm can be complicated.

**Strategy 3.**

| Operation | Comment |
|---|---|
| Input: $x_b$, $y$ | As in Strategy 1 |
| Distribute $x_b$, $y$ | $y$ according to the free distribution |
| $\chi = 0$ | As in Strategy 1 |
| Do until convergence | As in Strategy 1 |
| Distribute $\chi$ | As in Strategy 1 |
| $J = \chi^*\chi$, $\nabla_\chi J = \chi$ | As in Strategy 1 |
| $\Delta x = Q^{-1}\chi$ | As in Strategy 1 |
| $x = Mx_b + T\Delta x$ | As in Strategy 1 |
| For i=$n_t$,1,-1 | As in Strategy 1 |
| Get $x_i$ | As in Strategy 1 |
| Remote-copy $x_i \Rightarrow \hat{x}_i$ | According to the free distribution |
| $w_i = H_i \hat{x}_i - y_i$ | Local operation |
| $J = J + w_i^* O^{-1} w_i$ | Local operation if $O$ diagonal |
| $\hat{z}_i = H_i^* O^{-1} w_i$ | Local operation if $O$ diagonal |
| Copy-back $\hat{z}_i \Rightarrow z_i$ | According to the model distribution |
| $q_i = q_i + z_i$ | Add contribution, local operation |
| $q_{i-1} = T_{i,i-1}^* q_i$ | As in Strategy 1 |
| End For | |
| $s_0 = (Q^{-1})^* q_0$ | As in Strategy 1 |
| $\nabla_\chi J = \nabla_\chi J + s_0$ | As in Strategy 1 |
| Undistribute $\nabla_\chi J$ and $J$ | As in Strategy 1 |
| Call minimization routine | As in Strategy 1 |
| End Do | |

In this third alternative we do not have to re-distribute all the grid-point variables. We use only those variables near the observation points, $\hat{x}_i$ and $\hat{z}_i$, and distribute them equally, balancing the work load in the interpolation and integration parts of $H_i$. The disadvantage is an introduction of data structures for book-keeping of $\hat{x}_i$ and $\hat{z}_i$.

We can formulate a variant of this strategy where all horizontal dependencies are taken care of in the model space and where in the observation space only vertical dependencies are involved. The data structures and calculations are more easily structured in this way. Thus, we split the operator $H_i$ in two parts: (1) Interpolate horizontally in the model space without re-distributions of any data. (2) In the free distribution, take care of vertical dependencies and integrate to observed quantities. This reduces the communication by a factor of 4 for bi-linear interpolation and by a factor of 16 for bi-cubic interpolation. The drawback is of course the re-introduced work load imbalance due to the horizontal interpolations in the model space distribution. This imbalance is not severe and is compensated by the reduced communication overhead.

To conclude, we point out that implementations of the strategies on various computer architectures will perform differently and that in an operational code the efficiency is very important and hence the load balance must be good and the communication overhead low. Of the suggested strategies, alternative three seems to have most potential to be efficient on a computer with fast communication (for example Cray T3D). In this strategy the work load is balanced with a minimal amount of communication. In the alternative one the work load is not balanced and thus no extra communication is needed. This strategy would then be the best choice on a computer with large communication overhead (cluster of workstations). Alternative two, with equal work load on connected domains, seems to be preferable on shared memory machines. On a shared memory machine, there are no costs for data re-distributions. However, it is necessary to group data in order to avoid memory conflicts.

### 3.3 Analysis

In this section we will compare the suggested data distribution strategies in terms of execution time. We will use the following notation.

**Problem size:**

| | |
|---|---|
| $N_x$, $N_y$, $N_z$ | Grid resolution |
| $n_v$ | Number of analyzed variables |
| $N$ | $N_x \cdot N_y \cdot N_z \cdot n_v$ |

**Observations:**

| | |
|---|---|
| $Nobs_q$ | Number of observations of type $q$ |
| $Nobs_{q(i)}^{model}$, $Nobs_{q(i)}^{geo}$, $Nobs_{q(i)}^{free}$ | $Nobs_q$ in processor $i$ in the case of model, geographic and free distribution, respectively |
| $n_q$ | Number of data items in observation type $q$ |
| $n_t$ | Number of observation sets, each set corresponds to a certain point in time |
| $w_q$ | Arithmetic work to integrate observation type q |
| $w_{ip}$ | Arithmetic work of the interpolation formula |
| $N_{ip}$ | Width of the interpolation formula |

**Computer characteristics:**

| | |
|---|---|
| $p$ | Number of processors |
| $\alpha$ | Communication start up time |

| | |
|---|---|
| $\beta$ | Message transfer time per floating point number |
| $t_f$ | Time for a floating point operation |
| **Subroutine timings:** | |
| $t_{Q^{-1}}$, $t_{(Q^{-1})^*}$ | Variable transformations |
| $t_M$, $t_T$, $t_{T_i^*}$ | Model integrations |
| $t_{fmin}$ | Minimization routine |
| $t_{re2}$, $t_{re3}$ | Re-distributions in Strategy 2 and 3, respectively |

The algorithms can now be analyzed with simplified models, i.e. we disregard memory management, pipelining, message routing, synchronization, etc., and concentrate only on the arithmetic work (load balance) and the interprocessor communication. We estimate the CPU-time by counting the operations and the processor to processor messages. In the case of local operations we take the CPU-time of the most loaded processor.

**Strategy 1.**

| Operation | CPU-time |
|---|---|
| Distribute $x_b$ and $y$ | $\sum_{i=2}^{p}\left(\alpha+\beta\left[\sum_q Nobs_{q(i)}^{model}\cdot n_q+\frac{N}{p}\right]\right)$ |
| Distribute $\chi$ | $(p-1)\cdot(\alpha+\beta\frac{N}{p})$ |
| $J=\chi^*\chi$ | $\frac{N}{p}2t_f$ |
| $\Delta x = Q^{-1}\chi$ | $t_{Q^{-1}}$ |
| $x = Mx_b + T\Delta x$ | $t_M + t_T$ |
| Update overlaps | $4\left(\alpha+\beta\frac{N}{N_x}\right)$ |
| $w_i = H_i x_i - y_i$ | $\max_{1\le i\le p}\left(\sum_q Nobs_{q(i)}^{model}\left[w_{ip}+w_q+1\right]t_f\right)$ |
| $J = J + w_i^* O^{-1} w_i$ | $\max_{1\le i\le p}\left(\sum_q Nobs_{q(i)}^{model} 3t_f\right)$ |
| $q_i = q_i + H_i^* O^{-1} w_i$ | $\max_{1\le i\le p}\left(\sum_q Nobs_{q(i)}^{model}\left[w_{ip}^*+w_q^*+2\right]t_f\right)$ |
| $q_{i-1} = T_{i,i-1}^* q_i$ | $t_{T_i^*}$ |
| $s_0 = (Q^{-1})^* q_0$ | $t_{(Q^{-1})^*}$ |
| $\nabla_\chi J = \nabla_\chi J + s_0$ | $\frac{N}{p}t_f$ |
| Undistribute $\nabla_\chi J$, $J$ | $(p-1)(\alpha+\beta\frac{N}{p})+(p-1)(\alpha+\beta+t_f)$ |
| Call min routine | $t_{fmin}$ |

**Strategy 2.**

| Operation | CPU-time |
|---|---|
| Distribute $x_b$ and $y$ | $\sum_{i=2}^{p}\left(\alpha+\beta\left[\sum_q Nobs_{q(i)}^{geo}\cdot n_q+\frac{N}{p}\right]\right)$ |
| Distribute $\chi$ | As in Strategy 1 |
| $J=\chi^*\chi$ | As in Strategy 1 |
| $\Delta x = Q^{-1}\chi$ | As in Strategy 1 |
| $x = Mx_b + T\Delta x$ | As in Strategy 1 |
| Re-distribute $x_i$ | $t_{re2}$ |
| $w_i = H_i x_i - y_i$ | $\frac{1}{p}\sum_q Nobs_q\left(w_{ip}+w_q+1\right)t_f$ |
| $J = J + w_i^* O^{-1} w_i$ | $\max_{1\le i\le p}\left(\sum_q Nobs_{q(i)}^{geo}\cdot 3t_f\right)$ |
| $q_i = q_i + H_i^* O^{-1} w_i$ | $\frac{1}{p}\sum_q Nobs_q\left(w_{ip}^*+w_q^*+2\right)t_f$ |
| Re-distribute $q_i$ | $t_{re2}$ |

| | |
|---|---|
| $q_{i-1} = T^*_{i,i-1} q_i$ | As in Strategy 1 |
| $s_0 = (Q^{-1})^* q_0$ | As in Strategy 1 |
| $\nabla_\chi J = \nabla_\chi J + s_0$ | As in Strategy 1 |
| Undistribute $\nabla_\chi J$, $J$ | As in Strategy 1 |
| Call min routine | As in Strategy 1 |

The third strategy is identical to the second strategy except for the re-distributions. In the alternative formulation of Strategy 3 we do the horizontal part of the work associated with $w_{ip}$ in the model space distribution and only re-distribute these horizontally interpolated points.

Most the operations are equal in the suggested strategies. It is then illustrative to look at the time differences per step in the innermost loop, associated with $n_t$.

Strategy 1 vs. Strategy 2:

$$t_{s1} - t_{s2} \approx 4\left(\alpha + \beta \frac{N}{N_x}\right) +$$
$$\sum_{i=2}^{p}\left(\alpha + \beta\left[\sum_q Nobs^{model}_{q(i)} \cdot n_q\right]\right) - \sum_{i=2}^{p}\left(\alpha + \beta\left[\sum_q Nobs^{geo}_{q(i)} \cdot n_q\right]\right) +$$
$$\max_{1\le i\le p}\left(\sum_q Nobs^{model}_{q(i)}\left[w_{ip} + w_q + w^*_{ip} + w^*_q + 6\right] t_f\right) -$$
$$\frac{1}{p}\sum_q Nobs_q\left(w_{ip} + w_q + w^*_{ip} + w^*_q + 3\right) t_f - \max_{1\le i\le p}\left(\sum_q Nobs^{geo}_{q(i)} \cdot 3t_f\right) - 2t_{re2} \tag{11}$$

Total update of the overlaps is unnecessary and very costly. Only if we have an observation at the partition boundary we need to update points in the overlap. With the reduced update the first term in Equation (11) becomes negligible.

Strategy 2 vs. Strategy 3:

$$t_{s2} - t_{s3} \approx 2\left(t_{re2} - t_{re3}\right) \tag{12}$$

Here we have assumed that the number of observations are approximately the same in the two distributions, ($Nobs^{geo}_{q(i)} \approx Nobs^{free}_{q(i)}$). The critical term in the two comparisons is the CPU-time for re-distribution $t_{re}$, see Figure 1. We can model this by introducing a few more parameters.

**Re-distribution:**

| | |
|---|---|
| $p_c$ | Level of parallelism in the communication |
| $f_i$ | Fraction of data already in processor $i$ |
| $s_i$ | The number of processors data are spread to from processor $i$ |

With this notation we can roughly estimate the CPU-time for re-distribution:

$$t_{re2} \approx \frac{1}{p_c}\sum_{i=1}^{p}\left[(1 - f_i)\,\beta\frac{N}{p} + s_i\alpha\right] \tag{13}$$

and

$$t_{re3} \approx \frac{1}{p_c} \sum_{i=1}^{p} \left[ (1 - f_i)\beta \cdot \sum_{q} Nobs_{q(i)}^{model} N_{ip} n_v + s_i \alpha \right] \tag{14}$$

The difficult part is to estimate how much of the communication we can do in parallel, i.e. to estimate $p_c$.

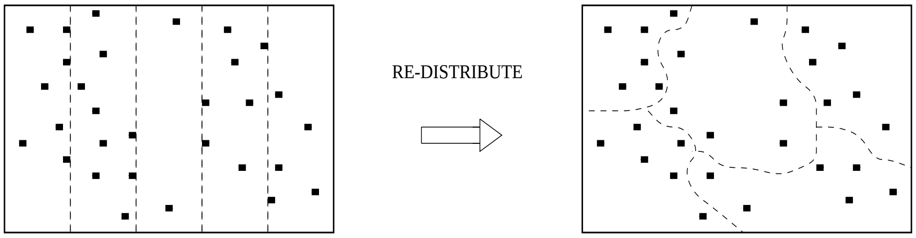



Figure 1: Re-distribution of $z_i$

### 3.4 A case study

With the help of equations (11) and (12) we can compare the three parallelization strategies in terms of CPU-time. We construct a hypothetical case and model the execution time first on a computer with fast communication (Cray T3D). It is obvious that Strategy 3 will give the best result. The purpose of the analysis is to estimate the gain in using this strategy.

**Problem size:**

$N_x = 180$, $N_y = 160$, $N_z = 16$

$n_v = 5$

**Observations:**

$Nobs_1 = 500$, $Nobs_2 = 1000$

$\max_i Nobs_{q(i)}^{model} = 0.1 \cdot Nobs_q$ , $Nobs_{q(i)}^{geo} = \frac{1}{p} Nobs_q$

$w_1 = 5$, $w_2 = 500$

$N_{ip} = 4 \cdot 16$, $w_{ip} = 2 \cdot N_{ip}$

**Computer characteristics:**

$p = 64$

$\alpha = 1 \mu s$

$\beta = 12 ns$

$t_f = 14 ns$ *peak/2*

**Subroutine timings:**

$t_{Q^{-1}} = 10ms$, $t_{(Q^{-1})^*} = 10ms$

$t_{M_i} = 10s$, $t_{T_i} = 15s$, $t_{T_i^*} = 15s$

$t_{fmin} = 0.1s$

**Re-distribution:**

$p_c = \frac{p}{2}$ *Half of the processors receive the data*

$\bar{f}_i = 0.5$ *Rough estimate*

$\bar{s}_i = 5$ *Rough estimate*

The estimation of the CPU-times for re-distribution becomes

$$t_{re2} \approx \frac{1}{32} \cdot 64 \left[ (1 - 0.5) \cdot 12 \cdot 10^{-9} \cdot \frac{2.3 \cdot 10^6}{64} + 5 \cdot 10^{-6} \right] = 442 \mu s$$

and

$$t_{re3} \approx \frac{1}{32} \cdot 64 \left[ (1 - 0.5) \cdot 12 \cdot 10^{-9} \cdot \frac{1500 \cdot 64 \cdot 5}{64} + 5 \cdot 10^{-6} \right] = 100 \mu s$$

In equation (11) the third and the fourth term cancel as the number of distributed observations are approximately the same. If we neglect the first term, by reduced update of overlaps, we have the time differences:

$$t_{s1} - t_{s2} \approx (0.1 \cdot 500 \cdot 272 + 0.1 \cdot 1000 \cdot 1262) \cdot 14 \cdot 10^{-9} -$$

$$\frac{1}{64} \cdot (500 \cdot 269 + 1000 \cdot 1259) \cdot 14 \cdot 10^{-9} - \frac{1500}{64} \cdot 3 \cdot 14 \cdot 10^{-9} - 2 \cdot 442 \cdot 10^{-6}$$

$$= 1957 \mu s - (305 + 884) \mu s = 768 \mu s$$

and

$$t_{s2} - t_{s3} \approx 2 \cdot (442 - 100) \mu s = 684 \mu s$$

The theoretical results are summarized in Table 1. We can see from the calculations above that the load imbalance in Strategy 1 and the re-distribution in Strategy 2 are the dominating factors. Total re-distribution is too costly, thus to save time we must only distribute the necessary data items, Strategy 3.

| *Algorithm* | $\Delta t$ ($\mu s$) |
|---|---:|
| Strategy 1 | 0 |
| Strategy 2 | -768 |
| Strategy 3 | -1452 |

Table 1: Relative theoretical timings of the parallelization strategies. Case Study 1, low communication overhead.

If we compare the strategies on a computer with a large communication overhead (workstation cluster), the Strategy 1 will be the best choice. If we take the same case as above but change the communication parameters to $\alpha = 100 \mu s$ and $\beta = 0.2 \mu s$ and redo the calculations above, we see that the communication overhead in Strategy 2 and 3 is larger than the load imbalance in Strategy 1. The results of the calculations are summarized in Table 2.

| *Algorithm* | $\Delta t$ ($\mu s$) |
|---|---:|
| Strategy 1 | 0 |
| Strategy 2 | 14700 |
| Strategy 3 | 3350 |

Table 2: Relative theoretical timings of the parallelization strategies. Case Study 2, large communication overhead.

Finally, on a shared memory computer, ($\alpha = 0$ and $\beta = 0$), the analysis above will show that the strategies two and three give the best results. In practice, Strategy 2 will perform better than Strategy 3 as the data points in the partitions are gathered in Strategy 2 and scattered in Strategy 3. The scattered distribution of data can cause memory conflicts and degrade the performance.

# 4 Partitioning algorithms

The analysis above did not consider the precise shapes of the partitions. In this section, we discuss different ways of partitioning the observation data. The question is: how do we partition the data in an optimal way, minimizing the data transfer in the re-distribution and balancing the work load in the calculations? This can be formulated as an optimization problem:

$$\textit{Minimize the cost function } f = max_p \left( C_{calc}^{(p)} + C_{redist}^{(p)} \right)$$

where $C_{calc}^{(p)}$ is the work load on node $p$ and $C_{redist}^{(p)}$ is the re-distribution time. Generally this is a too complex problem to solve exactly. The problem size is usually very large and it is difficult to create a correct and accurate cost function. Even the optimization problem above is a simplification of the global optimization of minimizing the overall execution time.

A number of approximate solution techniques exist for similar partitioning problems in computations with unstructured grids. The ones most commonly used are the recursive bisection methods [3], [12], and the greedy algorithm [10]. A class of iterative improvement methods [3] are used if an initial distribution is available as for example in adaptive grid computations with dynamic load balancing. Combinations of these two partitioning strategies are also common.

None of these methods are directly applicable to our case in the sense that they would give an approximate optimal solution to the optimization problem. The methods are usually adapted to problems with the data items connected resulting in partitions of connected domains. This is not necessary for our problem as the observations are not connected (we disregard satellite observations). For us the most important is to minimize the re-distribution time and to balance the work load. The iterative improvement methods above presume that the initial distribution is good and that only small changes have to be done. In our case the different data distributions (the model distribution and the observation distributions) can be quite different and the iterative methods would converge too slowly.

Nevertheless, we can try to implement the recursive coordinate bisection method as a first step. This method does not directly utilize that the data items are connected. It balances the work load by bisecting the computational domain alternatively in the $x$ and $y$ coordinate. As the data distribution in the model routines splits the domain in the $x$ (and $y$) direction the method has a potential of giving a good load balance while minimizing the re-distribution time but does not guarantee this if the observations are very unevenly distributed. This algorithm is also very fast and easy to implement. It is suitable for our parallelization Strategy 2 as it gives rectangular domains i.e. the work load is balanced and no complicated data structures are needed. It can also be used in Strategy 3, but here we can find better methods. Such a method is the *Possessive Greedy* method. It is a new variant of the Greedy algorithm. It is suited for the parallelization Strategy 3 as it clusters the observations more randomly. This method always guarantees a near optimal partitioning if we have an appropriate cost function. But, it is expensive compared to the recursive coordinate bisection method. Finally, as the number of observations varies in time, we can reform the generated partitions between two time steps by using an iterative improvement method.

### 4.1 Recursive Bisection with Iterative Improvement

It is important that we try to minimize the re-distribution cost while we balance the work load. In the model data distribution we have slices (spectral model). We could then alter the bisectioning method to only split in the $x$-direction to get slices also in the new data distribution. We sort the observations after their $x$-coordinate and assign a work factor to each observation proportional to the arithmetic work to integrate the data associated to the observation. The list of observations is then split recursively in two halves with equal work load. The computational domain is split accordingly to the $x$-coordinate of the observation list. We can expect to get data partitions that overlap with the model distribution at the ends of the domain but in the middle the partition slices can be totally displaced between the two data distributions.

We can get an improved algorithm by modifying the standard method, taking the re-distribution cost into account as well. In each bisection step we can use an iterative improvement algorithm. From equation (11) and (13) we can construct a cost function:

$$\begin{aligned} C_l &= \sum_q Nobs_{q(l)}^{geo} \cdot w_q \cdot t_f + t_{re} \\ C_r &= \sum_q Nobs_{q(r)}^{geo} \cdot w_q \cdot t_f + t_{re} \\ f &= \gamma \cdot \mathrm{abs}(C_l - C_r) + (1-\gamma) \cdot \max(C_l, C_r) \end{aligned} \tag{15}$$

where $C_l$ and $C_r$ are the total CPU-times of the left partition and of the right partition, respectively. Gamma is a weight factor. With (15) we can iteratively see how much we gain if we move the bisection line to the left or to the right and then do the bisection for the highest gain. By experimentally tuning the parameter gamma we can in some cases further improve the partitions. The second term in the cost function strives to minimize the computation time in each split. Sometimes this is not optimal. As the evaluation of the cost function is local in each step it can in the end lead to a very unbalanced partitioning. In contrary, the first term in the cost function tries to equalize the work load in each bisection step. With only this term present the computation time can be very high. To sum up, we can reduce the overall time by also considering the work balance in each step.

The recursive bisection method creates $2^r$ partitions where $r$ is an integer. We are then limited to using a number of processors that is a power of two.

The remedy is to also split the processors in two halves by integer division and then scale $C_l$ and $C_r$ accordingly. A detailed description of the algorithm can be found in Appendix A.1. Figure 2 shows an example of the type of result generated by this partitioning algorithm.

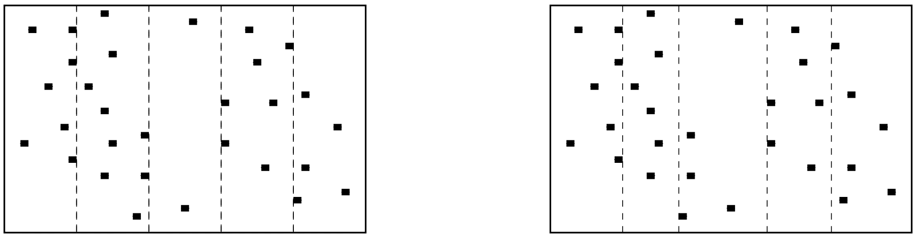


Figure 2: Model distribution (left) and Recursive Bisection distribution (right)

### 4.2 The Possessive Greedy Method

The Greedy method presumes that the work load in the most heavily loaded processor in the optimal partition lies somewhere between $W_T/p$ and $W_T/p+C$, where $W_T$ is the total arithmetic work. The algorithm makes a binary search to find the smallest constant $C$ that satisfies $w_i \leq W_T/p + C$ for the work load on each processor. For each test value of $C$ the algorithm tries to fill the processors with a work load that satisfies this condition and returns then a *true* if it was possible, else a *false*. In [10] an extra condition, that neighboring data items are to be placed in the same subset or in neighboring subsets, is used in the work assignment. As we do not have any connections between the data items, i.e. the observations (not satellite data), we can change this extra condition and adapt the method to our application.

To do this, we construct a *Probe* function that assigns the observations to the processors in two rounds. In the first round, only observations according to the model distribution are assigned to the processors. Depending on the work limit parameter $C$ we may not be able to assign all the observations, in the assignments we take both the arithmetic work and the communication cost into account. In the second round, the remaining observations are placed with a first fit algorithm in the incomplete processors. The algorithm can be found in detail in Appendix A.2. We name this new variant of the Greedy method the *Possessive Greedy* method as in the first round it tries to keep as many observations as possible from the model distribution. A result of the partitioning could look like in Figure 3. The principles of the method are independent of the shape of the model distribution. The model distribution could be arranged in blocks or in any other dividing.

This strategy guarantees minimal amount of data to be transfered in the re-distribution but it does not guarantee neighbor to neighbor communication. Here we assume that the cost to send data is approximately the same for any two processors in the machine. The algorithm can be improved further by, instead of first fit, trying to cluster the re-distributed observations in order to minimize the number of packages, i.e. the parameter $s_i$ in (14). This can be achieved by sorting the unassigned observations in round two.

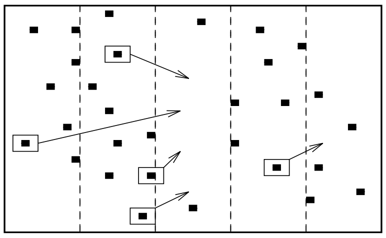


Figure 3: Distribution with Possessive Greedy, the model distribution is shown in the background and observations differing from this distribution are marked to which processor they belong in the new distribution.

# 5 Experiments

In this section we will examine the behavior of the suggested partitioning algorithms and evaluate their performance. The focus will be on distributed memory machines and Strategy 3, for Strategy 2 the re-distribution cost is very expensive so we can not expect any major modifications of the model distribution. The CPU-time is modeled theoretically, see Appendix B. We use real observation data provided by the Swedish Meteorological and Hydrological Institute (SMHI), see Figure 4. The observation file contains 2611 observations of five different types.

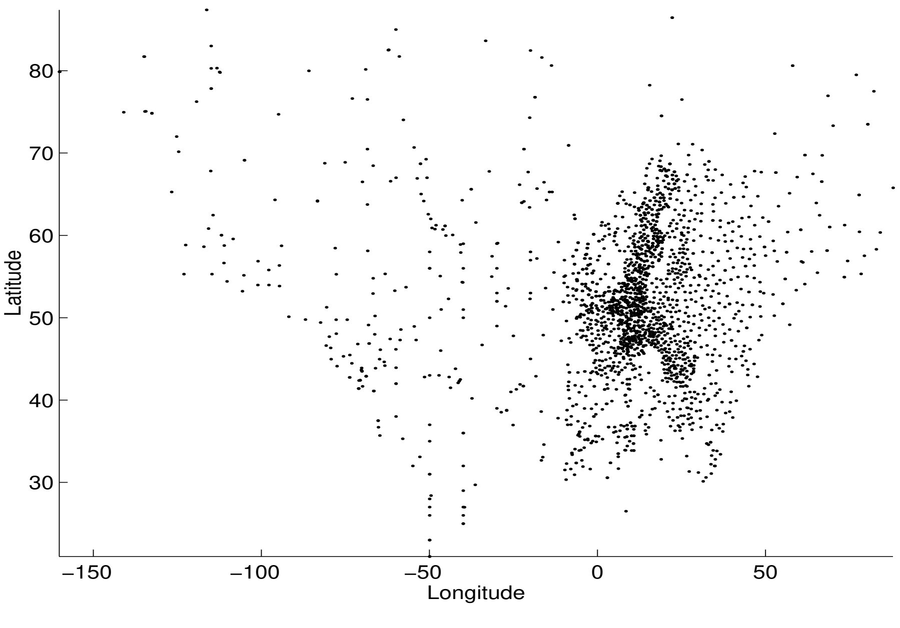
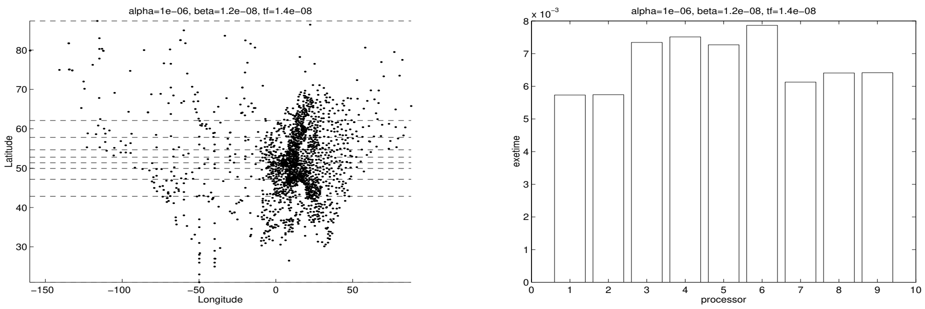



Figure 4: Distribution of observations used in data assimilation

## 5.1 Partitioning with the Recursive Bisection method

As a first experiment let us fix the problem size to 180x160x16 grid-points and five variables in each grid-point, the startup time $\alpha$ to $1.0\mu s$, the floating point number transfer time $\beta$ to $12ns$, the time for a floating point operation $t_f$ to

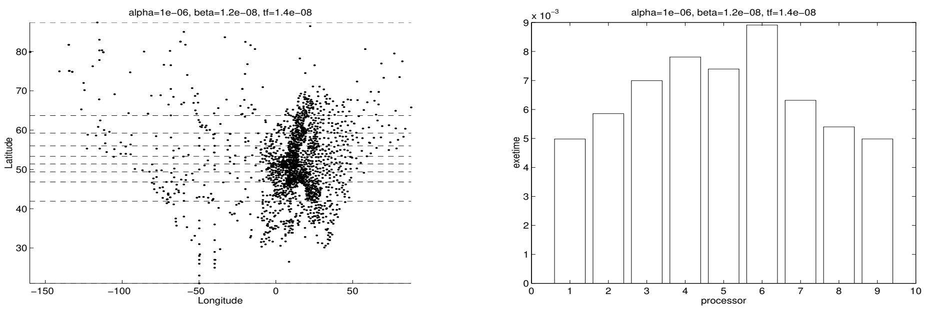
$14ns$, the cost parameter $\gamma$ to zero, and the number of processor to nine, see Figure 5. (The grid-points in the model distribution are partitioned in slices in the $y$-direction.)




(a) Observation distribution (b) CPU-time

Figure 5: Recursive Bisection with Iterative Improvement, Strategy 3

This result should be compared to the standard Recursive Bisection Method and to Strategy 1, Figures 6 and 7. The impact of $\gamma$ is negligible in this case as the algorithm manages to balance the work load well. Further tuning of $\gamma$ only marginally improves the partitions.



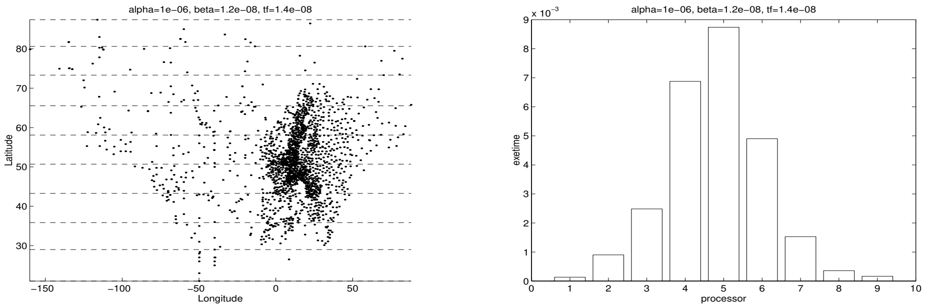

(a) Observation distribution (b) CPU-time

Figure 6: Standard Recursive Bisection Method, Strategy 3

We can see that the observations are tightly clustered in a small area. This results in totally displaced and resized partition shapes as compared to the model distribution. Thus, Strategy 1 without communication becomes competitive with Strategy 3 in combination with the Recursive Bisection method.

## 5.2 Partitioning with the Possessive Greedy method

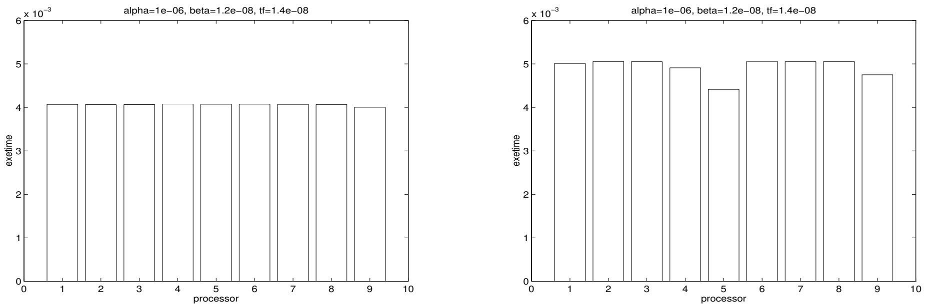
The Possessive Greedy method takes better advantage of the model distribution than the Recursive Bisection method. It also makes a global optimization over the processors. The prerequisites are much better than for the Recursive Bisection method. If we take the same problem as before we see, in Figure 8(a),




(a) Observation distribution (b) CPU-time

Figure 7: Observation and grid-point distribution equal, Strategy 1

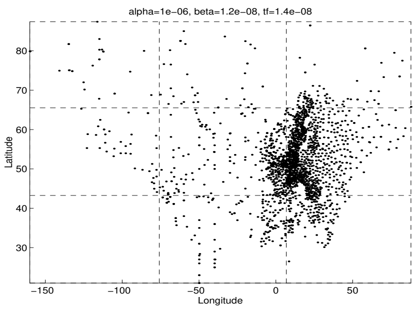
that we get a more optimal data distribution with Greedy than above. The Possessive Greedy method outperforms the previous methods. For the current set of observation points, it is about 20% better than the standard Greedy method, see Figure 8(b). Moreover, it is general enough to work on other model distributions and it does not require any tuning parameters.



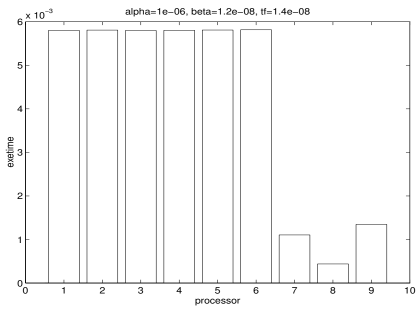

(a) Possessive Greedy (b) Standard Greedy

Figure 8: Greedy methods, Strategy 3, model distribution in slices

If the grid-points are partitioned in blocks more observations will be clustered in specific blocks in the model distribution. It will then require more communication to balance the work load and the overall performance will drop. We can see in Figure 9(a) that processor 6 holds most of the observations. The performance is limited by this processor which becomes overloaded due to communication. The situation can be slightly improved by rotating the coordinate system. The observation distribution becomes more homogeneous, since the empty corners will disappear.




(a) Model distribution




(b) CPU-time

Figure 9: The Possessive Greedy method, model distribution in blocks

The block-effect becomes more significant as we use more processors. Consequently the model distribution with slices will scale much better than the model distribution with blocks as the number of processors is increased, see Figure 10. Here the *speedup* is defined as the time to solve a fixed problem on one processor divided by the time to solve the same problem on $p$ processors. The timings are computed theoretically with a simplified model. The model does not consider any synchronization or collisions of messages. Thus, the plotted curves are the best we can expect.

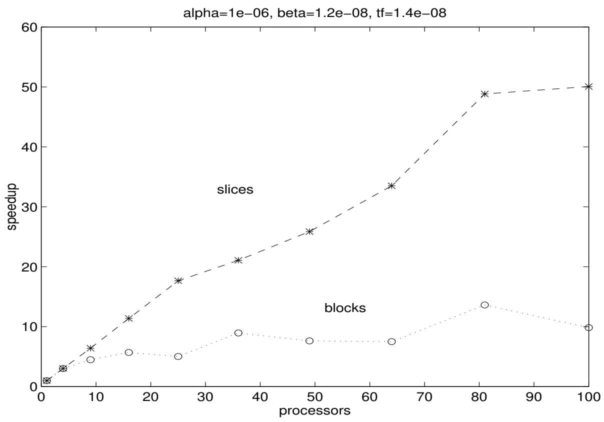



Figure 10: Theoretical parallel performance, Strategy 3, Possessive Greedy

# 6 Conclusions

We have investigated the prospects to parallelize a variational data assimilation scheme. The theoretical results show that parallelization of the observation integration is only a small part of the optimization scheme. The time savings in Table 1 are relatively small compared to the CPU-time spent in the model integration routines. The gain in implementing these partitioning algorithms in a 4D variational assimilation scheme will consequently be small. If we consider a 3D variational scheme the situation is different. Then, we do not have to integrate the model in time and the time savings become significant. A 3D variational scheme is the first step in the development of a full 4D variational assimilation scheme and can by itself be used in operational forecasting. Moreover, the data assimilation techniques are not only limited to weather forecasting and consequently the partitioning algorithms also have other areas of application. These techniques can be used in a number of other sciences. Examples of application areas are signal filtering, dynamical oceanography, biosphere modeling, navigation of aircraft and spacecraft, plasma physics and solid earth geophysics where we want to determine the internal state from observations at the surface. The dynamic models are here in some cases less demanding than in meteorology.

Moreover, we have suggested three data parallel strategies to do an implementation of the 4D variational assimilation scheme. Strategy 1 is to distribute the observations according to the model distribution. The second strategy is to re-distribute the grid-points to balance the work load while the third strategy is to have two data distributions, one for the grid-points and one for the observations. We have then given two partitioning algorithms to compute the data distributions, the Recursive Bisection method with Iterative Improvement and the Possessive Greedy method.

We have also made simulations of the different parallelizations. The results show that the Possessive Greedy method in combination with Strategy 3 is the best alternative for distributed memory machines with fast communication, for example Cray T3D(E). For computers with large communication overhead, for example cluster of work stations, Strategy 1 is best. On a shared memory machine we do not have any communication overhead, then it is enough to balance the work load. It is also advisable to use connected domains in order to avoid memory conflicts between different processors. Consequently, Strategy 2 would work better than Strategy 3 on a shared memory machine. Thus, the Recursive Bisection Method in combination with Strategy 2 is a good alternative on shared memory machines.

Finally, in the analysis and in the experiments we have omitted satellite data as the observation errors are not independent. It is important that the satellite soundings come into the algorithms as they can, with the variational techniques, be better utilized than before. This will be an interesting task for future work.

## Acknowledgments

We thank Michael Thuné[1], for his constructive comments and discussions on this work, and Nils Gustafsson[2] for sharing his expert knowledge.

[1] Department of Scientific Computing, Uppsala University
[2] Swedish Meteorological and Hydrological Institute

# References


[1] P. Courtier, *Variational Methods*, ECMWF, Reading, UK.

[2] R. Daley, *Atmospheric Data Analysis*, Cambridge University Press, 1991.

[3] J. De Keyser, D. Roose, *Load balancing data parallel programs on distributed memory computers*, Parallel Computing, 19:1199-1219, 1993.

[4] R. Fletcher, *Practical Methods of Optimization*, 2nd ed, Wiley, Chichester, 1987.

[5] J.C. Gilbert, C. Lemaréchal, *The modules M1QN3 and N1QN3*, INRIA, 1993.

[6] N. Gustafsson, D. Salmond, *A parallel spectral HIRLAM with the data parallel programming model and with message passing - a comparison*, Proceedings of the sixth ECMWF workshop on the use of parallel processors in meteorology, Reading, UK, November 21-25, 1994.

[7] N. Gustafsson, *The HIRLAM model*, ECMWF seminar proceedings, 9-13 September 1991.

[8] N. Gustafsson, X. Huang, *Sensitivity Experiments with the Spectral HIRLAM and its Adjoint*, Tellus, in press.

[9] N. Gustafsson, S. Hörnquist, *A 3-Dimensional Variational Data Assimilation (3DVAR) scheme for HIRLAM*, SMHI, Norrköping, Sweden, report in preparation.

[10] M.A. Iqbal, J.H. Saltz, S.H. Bohkari, *Performance tradeoffs in static and dynamic load balancing strategies*, Technical Report 86-13, ICASE, NASA Langley Research Center, Hampton, VA, 1986.

[11] L. Isaksen, S.R.M. Barros, *IFS 4D-Variational Analysis Overview and parallel strategy*, Proceedings of the sixth ECMWF workshop on the use of parallel processors in meteorology, Reading, UK, November 21-25, 1994.

[12] H.D. Simon, *Partitioning of unstructured problems for parallel processing*, Computing Systems in Engineering, 2(2/3):135-148, 1991.

[13] O. Talagrand, *Assimilation of observations, an introduction*, Laboratoire de Météorologie Dynamique du CNRS, Paris, France, 1995.

# A Partitioning algorithms

## A.1 The Recursive Bisection Algorithm

This algorithm is based on the *Recursive Coordinate Bisection* method. The method is modified to adapt to spectral model data distribution, i.e. slice distribution, and to our parallelization Strategy 2. The algorithm creates a new data distribution with consideration to work load and re-distribution time. The grid-point space is partitioned recursively in two halves by minimizing a cost function that regards both execution time and work load balance. The algorithm can be described in the following pseudo-code (we have omitted variable declarations, fixed parameters, etc.):

```
   recursive subroutine part( rcb_dist, jmin, jmax, pmin, pmax,       &
      obsmin, obsmax )
      in:   jmin, jmax, pmin, pmax, obsmin, obsmax
      out:  rcb_dist

! - - End recursion
      if ( pmin==pmax ) then
         rcb_dist(pmin,1)=jmin; rcb_dist(pmin,2)=jmax
         return
      end if

! - - Split the processors
      p=pmin+(pmax-pmin+1)/2

! - - Find optimal split of grid-point space
      j=jmin; iobs=obsmin; wl=0
      cmin=cost( p, j, iobs, wl )
      obsopt=iobs; jopt=j
      do j = jmin+1, jmax
         c=cost( p, j, iobs, wl )
         if ( c<cmin ) then
            obsopt=iobs; jopt=j; cmin=c
         end if
      end do

! - - Distribute each half recursively
      call part( rcb_dist, jmin, jopt, pmin, p-1, obsmin, obsopt-1 )
      call part( rcb_dist, jopt, jmax, p, pmax, obsopt, obsmax )

      return
   end subroutine
```

The cost function updates the work load *wl* of the left side partition by assigning more observations to the left side as the partition boundary moves to the right. Moreover, it adds a re-distribution time by comparing the new split with the model distribution of the grid-point space.

```
function cost( p, j, iobs, wl )
   in:     p, j
   inout:  iobs, wl

! - - Compute work load
      do while ( obs[iobs] ⇒ left )
         iobs=iobs+1; wl=wl+obs[iobs].work
      end do
      cl=wl*tf; cr=(wtot-wl)*tf
      cl=cl/(p-pmin); cr=cr/(pmax-p+1)

! - - Add re-distribution time
      [nsend_l, nrecv_l, nsend_r, nrecv_r]=diff( rcb_dist, model_dist )
      if ( nsend_l > 0 ) cl=cl+alpha+nsend_l*beta
      ...

! - - Compute the cost
      cost=gamma*abs( cl-cr ) + (1-gamma)*max( cl, cr)

      return
   end function
```

The arithmetic work of the algorithm can be estimated to $O(\log_2 p \cdot ny)$ operations, where $ny$ is the number of grid-points in the split direction.

We can easily adapt the algorithm to our parallelization Strategy 3. Instead of splitting the grid-point space we split the observation array. We then get two distributions, one for the grid-points (the model distribution) and one for the observations (the rcb distribution). The arithmetic work, $O(\log_2 p \cdot nobs)$, then becomes independent of the grid.

## A.2 The Possessive Greedy Algorithm

The Possessive Greedy algorithm is based on a function *probe* which tries to assign a work load $w_i \leq w_{\max}$ to the processors and returns a *true* if it was possible, else a *false*. The method makes a binary search for the smallest constant $w_{\max}$ for which the probe-function gives a *true* and returns the computed partitioning.

```
subroutine greedy( obs_dist )
   out:  obs_dist

! - - Binary search
      wlo=Wtot/p; whi=Wtot/p+C
      do while ( whi-wlo > eps )
         w=(whi+wlo)/2
         fit=probe( obs_dist, w )
         if ( fit ) then whi=w else wlo=w
      end do
```

```
      return
   end subroutine
```

The initial constant $C$ can be found with the probe-function by increasing the work parameter until all observations fits in the partitions. The binary search for a more accurate value can then start.

We can now construct a new probe-function and adapt the algorithm to our application (Strategy 3) by also considering the re-distribution cost. The observations are assigned to the processors in two rounds. In the first round observations already belonging to a processor are assigned to that processor. The remaining observations are assigned in the second round.

**Definition 1** *The set of unassigned observations is $U$ and the set of observations assigned to processor* **i***, in the new distribution, is $Q_i$. The set of observations already lying in processor* **i** *in the model distribution is $M_i$.*

The algorithm in the *probe* function can now be written in the following symbolic pseudo-code:

**function** *probe*( Q, $w_{\max}$ )

! - - *Save distribution from a previous iteration*

$Q_p^{old} = Q_p \ , \ \ \forall p$

! - - *Initialize with total re-distribution ($N_{ip}$ interpolation width)*

$w_p = N_{ip} \cdot \text{size}(M_p) \cdot \beta \ , \ \ \forall p$

! - - *Round one*

**do** p = 1, nproc

j = 0; $Q_p = 0$

! - - *Go through all observations in $M_p$*

**do while** ( j < size($M_p$) )

Take $obs \in M_p \setminus Q_p$

$w_{temp} = w_p - N_{ip} \cdot \beta + obs.work \cdot t_f$

! - - *Try to assign observation*

**if** ( $w_{temp} \leq w_{\max}$ **or** $w_{temp} \leq w_p$ ) **then**

Assign $obs$ to $Q_p$

$w_p = w_{temp}$

**end if**

! - - *Next observation*

j=j+1

**end while**

**end do**

! - - *Check if all* $w_p \leq w_{\max}$

**if** $(\max(w_p) > w_{\max})$ **then**
 probe = **false**
 $Q_p = Q_p^{old} \;, \;\; \forall p$
 **return**
**end if**

! - - *Define the set of unassigned observations*

Let $U = \sum_p M_p \setminus Q_p$
**if** ( $U == \emptyset$ ) **then**
 probe = **true**
 **return**
**end if**

! - - *Round two*

**do** p = 1, nproc
 j = 0 ; $N_u$ = size($U$)

! - - *Go through all unassigned observations*

**do while** ( j < $N_u$ )
 Take $obs \in U$
 $w_{temp} = w_p + N_{ip} \cdot \beta + obs.work \cdot t_f$

! - - *Try to assign observation*

**if** ( $w_{temp} \leq w_{\max}$ ) **then**
 Assign $obs$ to $Q_p$
 $w_p = w_{temp}$
 Remove $obs$ from $U$

! - - *Check if all fit*

**if** ( $U == \emptyset$ ) **then**
 probe = **true**
 **return**
**end if**
**end if**

! - - *Next observation*

j=j+1
**end while**
**end do**

! - - *Algorithm failed to fit observations*

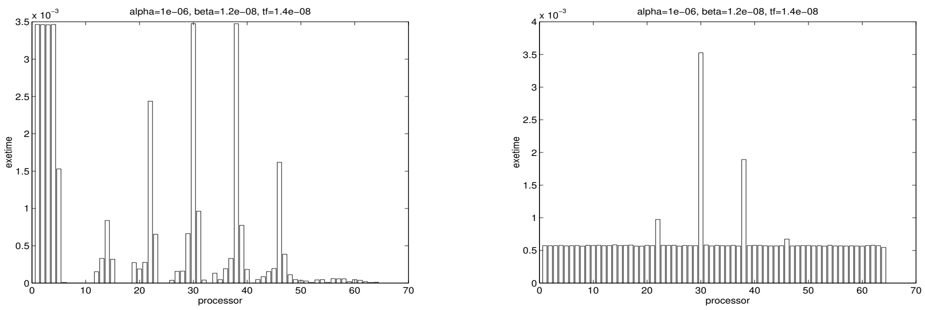
probe = **false**
$Q_p = Q_p^{old} \;, \;\; \forall p$
**return**
**end function**

In comparison with the bisectioning method this algorithm is general enough to directly work on any model distribution but it is more laborious.

**Proposition 1** *The arithmetic work of the Possessive Greedy method can be estimated to* $O(\log_2(C/eps) \cdot p \cdot nobs)$.

The code includes comments but some of the statements may require further illumination of why they are needed. In the algorithm we save the distribution from a previous iteration (the first statement). This assures that we have the last valid data distribution as the binary search is terminated. The next statement in the code initializes the work load with total re-distribution, i.e. no observations are in the new partitions. Observations are then gradually inserted into the partitions. If the communication is heavy the initialization can fill the work load and no observations can fit in the partition. To prevent this we allow observations in the partitions that decreases the work load, (the *or* part in observation assignment). We must later also check that the work load condition is not violated, (*check if* $w_p \leq w_{max}$).

As the complexity of the algorithm increases we can suggest more modifications to it. One modification is to remove the work load condition test mentioned above. The algorithm still gives a partition with the same work load on the most heavily loaded processor as before but it also tries to minimize and balance the work load on the other processors. This partition looks better for the eye but can in practice increase the re-distribution time as more communication is needed to balance the work load on the other processors, see Figure 11.




(a) Limited work load in round 1 (b) No work load test in round 1

Figure 11: Theoretical CPU-timings from two variants of the greedy algorithm

The algorithm neglects the latency in the communication. This can be compensated by increasing the value of the communication, i.e. increasing the parameter $\beta$. Numerical experiments verifies that this can improve the partitions. A third modification would be to sort the unassigned observations in round two. This would give more collected messages but it also would increase the partitioning time.

## B Theoretical CPU-timings

In the Figures 5 to 10 we have modeled the CPU-times theoretically. As we have the exact data distribution available we can let a program compute the CPU-time for each processor.

```
subroutine exetime( time, obs_dist, model_dist )
   in:   obs_dist, model_dist
   out:  time

! - - Arithmetic work load
   do p = 1, nproc
      do iobs = obs_dist(p,1), obs_dist(p,2)
         time(p) = time(p) + obs[iobs].work * tf
      end do
   end do

! - - Re-distribution time
   do p = 1, nproc
      do iobs = obs_dist(p,1), obs_dist(p,2)
         if ( iobs not in model_dist(p,1:2) ) then
            find p2, { iobs in model_dist(p2,1:2) }
         end if
         recv(p,p2)=recv(p,p2)+1; send(p2,p)=send(p2,p)+1
      end do
   end do
   if (send(i,j)>0) time(i)=time(i)+α+β · N_ip· send(i,j) ,   ∀i, j, i ≠ j
   if (recv(i,j)>0) time(i)=time(i)+α+β · N_ip· recv(i,j) ,   ∀i, j, i ≠ j

   return
end subroutine
```

We have not considered any restraints in the parameter $p_c$ in Equation (14), we assume that we can communicate without any delays. In practice the processors will have to wait for each other and there will be collisions of messages on the communication network.

This routine is not only useful for computing theoretical CPU-times, in connection with the computation of the data partitions we can use the above algorithm to initiate receive and send tables. These tables can then be used to collect data in long messages and to schedule the communication in the computations. That is, we can optimize the communication with help of the send and receive tables.