DAO Office Note 97-02

Office Note Series on Global Modeling and Data Assimilation

Richard B. Rood, Head
Data Assimilation Office
Goddard Space Flight Center
Greenbelt, Maryland

Parallel Implementation of a Kalman Filter for Constituent Data Assimilation

P. M. Lyster*, S. E. Cohn, R. Ménard**, L.-P. Chang†, S.-J. Lin**
R. G. Olsen‡

Data Assimilation Office, Goddard Laboratory for Atmospheres

Additional affiliations:
∗ Joint Center for Earth System Science
∗∗ Joint Center for Earth Systems Technology
† General Sciences Corporation (a subsidiary of
Science Applications International Corporation)
‡ Department of Mathematical Sciences,
Northern Illinois University, DeKalb, Illinois

Goddard Space Flight Center
Greenbelt, Maryland 20771
April 1997

Abstract

A Kalman filter for the assimilation of long-lived atmospheric chemical constituents was developed for two-dimensional transport models on isentropic surfaces over the globe. Since the Kalman filter calculates the error covariances of the estimated constituent field, there are five dimensions to this problem, $\mathbf{x_1}$, $\mathbf{x_2}$, and *time*, where $\mathbf{x_1}$ and $\mathbf{x_2}$ are the positions of two points on an isentropic surface. Only computers with large memory capacity and high floating point speed can handle problems of this magnitude. This article describes an implementation of the Kalman filter for distributed-memory, message-passing parallel computers. To evolve the forecast error covariance matrix, an Operator Decomposition and a Covariance Decomposition were studied. The latter was found to be scalable and has the general property, of considerable practical advantage, that the dynamical model does not need to be parallelized. Tests of the Kalman filter code examined variance transport and observability properties. This code is being used currently to assimilate constituent data retrieved by limb sounders on the Upper Atmosphere Research Satellite.

Published in *Mon. Wea. Rev.*, **125**, 1674-1686 (1997).
Available at `http://dao.gsfc.nasa.gov/subpages/office-notes.html`

Contents

1 Introduction

This article introduces one of the current research efforts at the Data Assimilation Office (DAO) of the NASA/Goddard Space Flight Center to use the Kalman filter (e.g., Ghil et al. 1981) for atmospheric data assimilation. At present, a full implementation of the Kalman filter in a four-dimensional data assimilation (4DDA) context is impossible. Considerable research needs to be undertaken before any implementation could be used operationally. Many open questions need to be answered surrounding computational approximations (e.g., Todling and Cohn 1994; Cohn and Todling 1996), model and observation error covariance descriptions (e.g., Dee 1995), nonlinearity (e.g., Ménard 1994), and basic probabilistic assumptions (e.g., Cohn 1997 and references therein). Therefore we have chosen a model problem in two space dimensions, for which real observational data exist and for which the Kalman filter can be implemented fully, to establish a benchmark system to begin addressing some of these issues in a real-data environment.

Our model problem focuses on trace chemical constituent assimilation. This is also a problem of considerable interest in the Earth Science community (e.g., Daley 1995, Riishøjgaard 1996). It is well known that in the upper troposphere and stratosphere, a number of trace chemical constituents can be modeled for relatively long timescales, typically weeks to months, using mass continuity dynamics. In isentropic vertical coordinates the transport behaves two-dimensionally. Therefore we have implemented a Kalman filter in spherical geometry on an arbitrary isentropic surface (*cf.* Cohn and Parrish 1991). In this case the state dimension is 1.3×10^4 at a resolution of 2^o latitude $\times$ 2.5^o longitude, which requires special computational strategies for a full Kalman filter implementation. Observations are available from the Upper Atmosphere Research Satellite (UARS) (Reber 1993; Rood and Geller 1994) launched in September 1991. This NASA satellite carries a number of instruments that obtain retrievals of trace gases in the upper troposphere and the stratosphere using limb-sounding techniques. Thus we can perform meaningful data-assimilation experiments that operate at the floating point speed and memory limit of present-generation distributed-memory parallel computers. This article deals with efficient strategies for parallel implementation of the Kalman filter, and tests their implementation by assessing basic scientific properties of variance transport and observability.

Since this article concentrates on computational aspects of full Kalman filter implementation, synthetic data are used in the experiments reported here. Near-future work will involve assimilating actual UARS data, with the transport model driven by wind analyses from the global atmospheric data assimilation system (PSAS; da Silva et al. 1995) currently under development at the DAO. With the benchmark constituent data assimilation system

in place, we expect to be able to address a number of the open questions in Kalman filtering, and to produce research-quality datasets of assimilated atmospheric constituents at an acceptable cost.

This paper is divided into six sections. Section 2 presents the mathematical formulation of the Kalman filter for constituent data assimilation. Section 3 describes the implementation on distributed-memory parallel computers using message-passing Fortran 77 software. We develop two methods for implementing the forecast error covariance dynamics and indicate our reasons for choosing one (Covariance Decomposition) over the other (Operator Decomposition). The Covariance Decomposition is efficient in the sense of minimizing wall-clock time, and scalable in the sense that speed up is attained when more processors are used on a given problem (especially at high resolution). It also has the important advantage that the model dynamics does not need to be parallelized so long as the model fits in the memory of a single processor of the parallel computer – this is a general property. We further describe a parallel implementation of the Kalman filter analysis equations. Section 4 emphasizes the efficiency of the parallel implementation by showing detailed timings on the 512-processor Intel Paragon computer at the California Institute of Technology. In section 5 we concentrate on the scientific validation of the algorithm itself by testing two basic properties of our Kalman filter algorithm. The first test verifies the predicted transport by solid-body rotation winds of an initial cosine-hill variance structure. The second test shows how the total variance is reduced to zero to machine precision in finite time for an observing network that guarantees complete observability. In section 6 we summarize our conclusions.

2 Description of the Kalman Filter for Constituent Assimilation

The transport of atmospheric chemical constituents obeys the mass conservation law:

$$\frac{\partial}{\partial t}\rho + \nabla \cdot \mathbf{v}\rho = S, \tag{1}$$

where ρ denotes the density of the constituent (i.e., its mass per unit volume), $\mathbf{v}$ is the three-dimensional wind vector, and S represents the mass source/sink terms due to chemical reactions or photodissociation.

In this work we consider the transport of long-lived constituents (i.e., chemical tracers). Lower and middle stratospheric nitrous oxide (N_2O), methane (CH_4), CFC's, water vapor, aerosols, and lower stratospheric ozone (O_3), can all be characterized as long-lived constituents for time scales of weeks or more (Brasseur and Solomon 1984, Andrews et al. 1987). Using potential temperature θ as the vertical coordinate, and neglecting diabatic

effects, chemistry, and explicit sub-grid scale parameterization of mass flux, the transport of long-lived constituents becomes two-dimensional, and can be written as

$$\frac{\partial}{\partial t}\rho + \nabla_\theta \cdot \mathbf{v}_\theta \rho = 0. \tag{2}$$

Here $\mathbf{v}_\theta$ denotes the two-dimensional wind vector on the isentropic surface ($\theta = constant$), and ∇_θ denotes the two-dimensional gradient operator on the isentropic surface. The mass conservation law can also be written in terms of mixing ratio instead of density as the state variable, in which case the appropriate transport model is the advection equation (*cf.* Andrews et al. 1987, Appendix 10A).

In studies of tracer transport, winds used to drive the transport model (1) or (2) are usually given by a general circulation model (Williamson and Rasch 1989) or from wind analyses interpolated in time (Rood et al. 1991). However, for this study we use analytically prescribed wind fields to assess basic properties of the Kalman filter algorithm as well as the timing and scaling performance of the parallel implementation.

In matrix-vector notation, a discrete version of Eq. (2) can be written as

$$w_k^t = M_{k-1} w_{k-1}^t, \tag{3}$$

where w_k^t is an n-vector of constituent densities on a grid covering the isentropic surface, and the $n \times n$ matrix M_{k-1} denotes the action of the discrete dynamics from time t_{k-1} to time t_k. The continuum transport equation (2) is linear and it is assumed that the discrete transport equation (3) is also linear; the dynamics matrix M_k does not depend on w_k^t, although it does depend on the wind field, which may vary with time. Two different discretizations were actually implemented, as discussed in section 3.2. For both cases, the discrete dynamics are assumed to be perfect in this initial study: no model error term appears in Eq. (3). Thus w_k^t denotes the true state at time t_k, which is to be estimated on the basis of observations available up to and including time t_k.

Observations w_k^o available at time t_k are assumed to have the form

$$w_k^o = H_k w_k^t + \varepsilon_k^o, \tag{4}$$

where w_k^o is a p-vector of observations valid at time t_k (p generally varies with time, $p = p_k$), H_k is the $p \times n$ observation matrix used to interpolate the state to the positions of the observations, and ε_k^o is a random vector representing the observational error, assumed to be white in time, Gaussian-distributed with zero mean and known covariance $R_k = < \varepsilon_k^o (\varepsilon_k^o)^T >$, and uncorrelated with the initial state w_0^t. H_k is assumed to be independent of the state w_k^t, and is implemented as a sparse operator performing bilinear interpolation from the model grid to the observation locations. The error of representativeness is neglected

here. For further discussion of model error and representativeness error, see Cohn (1997) and references therein.

Under the stated assumptions, the standard Kalman filter algorithm described below gives the evolution of the conditional means

$$w_k^f \equiv < w_k^t | w_1^o, w_2^o, ..., w_{k-1}^o >, \tag{5}$$

$$w_k^a \equiv < w_k^t | w_1^o, w_2^o, ..., w_k^o >, \tag{6}$$

and the corresponding conditional covariances

$$P_k^f \equiv < \varepsilon_k^f (\varepsilon_k^f)^T | w_1^o, w_2^o, ..., w_{k-1}^o >, \tag{7}$$

$$P_k^a \equiv < \varepsilon_k^a (\varepsilon_k^a)^T | w_1^o, w_2^o, ..., w_k^o > . \tag{8}$$

Here the n-vectors w_k^f and w_k^a are termed the forecast and analysis, respectively, $\varepsilon_k^f = w_k^t - w_k^f$ and $\varepsilon_k^a = w_k^t - w_k^a$ are the forecast and analysis errors, and P_k^f and P_k^a are the $(n \times n)$ forecast and analysis error covariance matrices.

The Kalman filter algorithm (see Jazwinski 1970, Gelb 1974, or Cohn 1997 for derivations) consists of two steps:

The forecast step

$$w_k^f = M_{k-1} w_{k-1}^a \tag{9}$$

$$P_k^f = M_{k-1} P_{k-1}^a {M_{k-1}}^T = M_{k-1} (M_{k-1} P_{k-1}^a)^T \tag{10}$$

The analysis step

$$w_k^a = w_k^f + K_k (w_k^o - H_k w_k^f) \tag{11}$$

$$K_k = P_k^f H_k^T (H_k P_k^f H_k^T + R_k)^{-1} \tag{12}$$

$$P_k^a = (I - K_k H_k) P_k^f (I - K_k H_k)^T + K_k R_k K_k^T . \tag{13}$$

The (unknown) initial true state w_0^t is assumed to be Gaussian-distributed, with known mean w_0^a and covariance matrix P_0^a. In the covariance evolution equation (10), the second equality is used because we implement the dynamics matrix M_k as an operator.

It should be noted that except for roundoff errors in the computation of Eq. (13), the analysis error covariance matrix is symmetric and positive semidefinite for any choice of gain matrix K_k. When the optimal Kalman gain (12) is used, the analysis error covariance equation simplifies to

$$P_k^a = (I - K_k H_k) P_k^f , \tag{14}$$

the optimal form of the analysis error covariance equation. While the optimal form involves less computation than the so-called Joseph form (13) with K_k given by Eq. (12), P_k^a computed using the Joseph form is less susceptible to roundoff errors in the evaluation of K_k (Bucy and Joseph 1968, pp. 174-176; Gelb 1974).

3 Implementation Strategies for Distributed-Memory Parallel Computers

The computation involved in the Kalman filter, especially in Eqs. (10), (13), or (14), is floating point count- and memory-intensive. To implement the Kalman filter we use recent advances in the use of distributed-memory parallel computers. Distribution strategies, their relative efficiencies, and details of the corresponding algorithms are discussed in this section, first for the forecast step and then for the analysis step.

The style of programming we have adopted is Single Program with Multiple Data (SPMD). This means that the same compiled program is run on all processors (SP), but each processor is responsible for different parts of the distributed memory (MD). Our code runs portably on serial machines, such as a single processor of a Cray C90 if it fits into memory, or on multi-processor message-passing distributed-memory computers; the distinction is made by setting the number of processors (a Fortran parameter) to be $N_p = 1$, or $N_p > 1$ respectively.

Our implementation to date has been on Intel parallel computers, specifically on the Paragon computer at the California Institute of Technology (Caltech), which has 512 processors and about 24 megabytes of usable memory per processor. We also used the Touchstone Delta at Caltech, an older machine with 512 processors and 12.5 megabytes of usable memory per processor. Typical processor speeds on both of these machines range from 2 to 20 million floating point operations per second (megaflop/s) per processor for realistic applications, thus reaching 1 to 10 gigaflop/s all told. For this paper we used the NX communications library; we used a modular programming approach so that the more standard Message Passing Interface (MPI) communications library can also be used.

3.1 Implementation of the covariance forecast, $M(MP)^T$

The computation of the covariance forecast, Eq. (10), represents one of the most computationally demanding parts of the Kalman filter algorithm. The dynamics matrix M is a sparse operator, occupying $O(n)$ words of memory; the components of M are generated from the wind variables(u, v) that are specified on a latitude-longitude grid. However, P is a full matrix with n^2 non-zero elements, which is a large memory burden for the computer. For example, at 2^o (latitude) $\times$ 2.5^o (longitude) resolution $n \approx 1.3 \times 10^4$, and this matrix represents about 168 megawords, or 670 megabytes for a single-precision (4 bytes per word) implementation. Thus the computation of $M(MP)^T$ involves not only floating point cost of about hn^2 per timestep, where h depends on the finite difference template for M (typically

$h \approx 50$), but also the memory cost of storage. The compiled code for the entire Kalman filter based on $2^o \times 2.5^o$ resolution fits easily in the memory of the Intel Paragon, but not on the Cray C90 at GSFC.

It follows that it is important to distribute effectively the large matrix P over the available processors. This should be done with minimal redundancy in order to conserve memory, and as uniformly as possible in order to balance the memory and computational load over the processors. We have considered two such strategies for this domain decomposition: Operator Decomposition; and Covariance Decomposition.

The Operator Decomposition follows naturally from the standard domain decomposition of a finite-difference model where all state-like vectors (w and columns of P) are individually partitioned and distributed among the processors. This can be used because the operation MP can be regarded as repeated actions of the model operator on state-like columns of P. The details of the resulting algorithm, described in the next paragraphs, show that the operation $M(MP)^T$ can be performed without the need for a global transpose of data amongst the processors. The Covariance Decomposition avoids the need to domain decompose the model by acting M on whole columns of P, i.e., P is domain decomposed by distributing whole columns of P among the processors. This is of great practical importance since any model can be used without having to develop a specialized model domain decomposition. This is a general property for parallel Kalman filters on large state spaces. The resulting algorithm for $M(MP)^T$ is forced to use a global transpose of the large matrix MP. The timings presented in the next section show that this is not deleterious to performance.

3.1.1 Operator Decomposition

We adopt the Fortran notation representing the state w on a latitude-longitude grid with indices $w(1{:}Nx, 0{:}Ny)$; the memory is aligned contiguously along rows starting at $w(1,0)$ and ending at $w(Nx, Ny)$, N_x being the number of grid points on each circle of latitude and $N_y + 1$ the number on each meridian. The square matrix $P(i1, j1, i2, j2)$ then has columns (not to be confused with the columns or rows of the state-like variables on the latitude-longitude grid) that extend from $P(1, 0, i2, j2)$ to $P(Nx, Ny, i2, j2)$, where the Fortran indices $(i2, j2)$ specify a particular column of P. The operation MP can be represented as $[MP_1, MP_2, ..., MP_i, ..., MP_n]$ where P_i is the ith column of P. These P_i's are state-like quantities with the same structure as w.

The operator decomposition is based on a decomposition of the domain of the transport operator M. For the state forecast, Eq. (9), this is a classical domain-decomposition algorithm (Foster 1995). For the covariance forecast, the algorithm is illustrated in Figure 1(a).

The dashed lines in the box representing P delineate the elements or slice of P that belong to a particular processor. When the grid-point transport model operates on this slice, only data pertaining to a fraction of the physical domain are needed. In this method, the domain of the transport operator is decomposed and the columns of the covariance matrix are decomposed accordingly. For a specific discretization of M, certain boundary values of a slice of P in each domain need to be stored redundantly in guard cells. If the number of grid points in each domain is large compared to the number in the boundary regions, this is a small degree of redundancy. However, the redundant data have to be passed between appropriate processors when M operates on a column P_i (or w). This is called message passing and it involves an interprocessor communication time cost that must be added to the on-processor floating point operation time cost when evaluating the wall-clock time, or more importantly, the feasibility of performing the algorithm in an acceptable amount of time. An advantage of this Operator Decomposition approach is that the transpose $(MP)^T$ involves no communications. As illustrated in Figure 1(a) the slice of MP in a particular processor is actually stored as a collection of column fragments. These data are rearranged in memory to form contiguous rows of MP. This is equivalent to forming a domain decomposition of $(MP)^T$ where whole columns are stored on each processor. When the entire two-dimensional wind field is in each processor, which is not a strain on memory, $M(MP)^T$ can be evaluated, without message passing, by the operator M acting on the columns of $(MP)^T$, i.e., evaluate $[M(MP)^T_1, M(MP)^T_2, ..., M(MP)^T_i, ..., M(MP)^T_n]$. Finally, because P is symmetric the columns of $M(MP)^T$ can be internally transposed so that the resulting matrix is domain decomposed, suitable for continuing the timestep cycle.

3.1.2 Covariance Decomposition

In this case, the error covariance matrix P is partitioned along rows so that whole columns are stored contiguously on each processor. The transport model operates on whole columns of P as illustrated in Figure 1(b). It is not trivial to partition a size-n^2 matrix P along rows onto N_p processors in such a way that the number of columns of P (and hence also the floating point cost of MP) is approximately the same on all processors. This is generally referred to as the problem of load balancing. On a message-passing computer with $N_p \gg 1$ it is acceptable for a relatively few processors to finish their jobs earlier than the rest; these processors just sit and wait. However, it is a problem if a relatively few processors finish much later than the rest. Lyster et al. (1997) describe the load balancing procedure that was applied to the Covariance Decomposition approach; the algorithm is summarized in Appendix A. The matrix MP is calculated with no interprocessor communications as long

as all the wind components are stored in each processor. The result MP is decomposed naturally in the same manner as P was as indicated in Fig. 1(b).

The transpose $(MP)^T$ has to be performed so that whole columns of the result will be stored contiguously in-processor, in preparation for the calculation of $M(MP)^T$ in the same manner as MP itself. This necessarily involves communications because blocks of MP that belong to a processor must be communicated to the destination processor that will store $(MP)^T$. This amounts to a global transpose of a size-n^2 matrix, which is not trivial since every processor must send and receive sub-blocks of P to every other processor. Efficient implementation of this global transpose using Intel NX communication library subroutines is also described in Lyster et al. (1997). After the global transpose, the complete calculation $M(MP)^T$ can be computed simply, without communications, in exactly the same way as the final step of the Operator Decomposition approach described above.

In both approaches the whole (symmetric) matrix P is stored. This is not wasteful of memory since both approaches calculate $M(MP)^T$ through intermediate calculation of the non-symmetric matrix $(MP)^T$ in the same memory as that allocated to P. Storing the whole of P also simplifies both algorithms considerably.

3.2 Comparison of the Operator and Covariance Decompositions

Comparing multiple approaches to an application is generally based on the nature of the software implementation (complexity, portability, ease of debugging and maintenance, etc.), and the relative efficiencies in terms of metrics such as the achievable number of floating point operations per second or the time to solution.

The relative efficiencies of the two decomposition approaches are determined by how much of the work can be distributed effectively (parallelized) and by how much the parallel cost of interprocessor communications and associated memory buffering detracts from the on-processor floating point operation performance. The on-processor floating point count is approximately the same in both cases. Also, not only is it important that the parallel cost be small, but that it remain relatively small as the number of processors N_p is increased. This is commonly referred to as scaling. In our work, it is important that an algorithm scales well for large numbers of processors (say $N_p \approx 500$) for typical resolutions of $4^o \times 5^o$ and $2^o \times 2.5^o$.

We used two different transport schemes for the operator M: the monotonic second-order upwind van Leer scheme (Rood 1987, Allen et al. 1991) and a flux-conserving semi-Lagrangian piecewise parabolic method (Lin and Rood 1996). We evaluated the Operator Decomposition algorithm only for the van Leer transport scheme. The conclusions that we

drew from this and the well-known difficulty of domain decomposition for semi-Lagrangian transport (e.g., Barros et al. 1995) led us to focus on the Covariance Decomposition.

First, we can estimate the central processing unit (CPU) time it takes to perform $M(MP)^T$ excluding the parallel cost. At $4^o \times 5^o$ resolution for the van Leer scheme on a single processor of the Intel Delta, the operation Mw takes 0.077 seconds per timestep. This time does not differ much from the time for the algorithm of Lin and Rood (1996). At this resolution $n = 72 \times 46 = 3312$, so the minimum time to evaluate $M(MP)^T$ is $(2n/N_p) \times 0.077 \approx 512/N_p$ seconds per timestep. A typical simulation uses a 15-minute timestep on 256 processors, so this amounts to 192 seconds of compute-time per day (96 timesteps). This establishes that an efficient parallel implementation of the dynamics should give rise to an algorithm that runs to completion in an acceptable amount of wall-clock time. A run at $2^o \times 2.5^o$ resolution with the same timestep (made possible because 15 minutes was conservative for the $4^o \times 5^o$ run) should take about $4^2 = 16$ times as long, since P is then four times as large in each dimension.

The scaling of the Operator Decomposition was assessed by developing a domain-decomposed version of the van Leer scheme for Eq. (2). This involved dividing the latitude-longitude grid uniformly into N_{px} regions in the E-W direction and N_{py} regions in the N-S direction (i.e., $N_p = N_{px} \times N_{py}$). It should be noted that this is not an optimal decomposition for this scheme because the standard upwind algorithm on a latitude-longitude grid usually requires subcycling of the timestep at high latitudes in order to keep the Courant number less than one. Hence this uniform domain decomposition is load-imbalanced because processors that solve for high-latitude domains have a higher CPU burden. To focus attention on scalability we do not directly address this load imbalance problem.

The results given here are for the case of a small timestep everywhere on the grid such that the Courant number is less than one, and therefore there is no load imbalance. The metric we use is the speedup S, which is the time taken to perform Mw (or MP_i) on one processor divided by the time on N_p processors. If there is no communication cost and a fixed processor speed we would expect an ideal scaling $S_{ideal} = N_p$. When only parallel communications degrade the scaling performance we expect a speedup of

$$S_c = N_p/(1 + \tau_{par}/\tau_{CPU}), \tag{15}$$

where τ_{par} is the time involved in packing and unpacking the communication buffers and invoking the communication library subroutines. The quantity τ_{CPU} is the processor time used for floating point operations. In general, maximum times per processor should be used for times such as τ_{par} and τ_{CPU}. However, here and for the remainder of this paper, where load balance is never a problem, we will use average times per processor.

Figure 2 shows a plot of the measured speedup S as well as the ideal speedup for a $4^o \times 5^o$ resolution problem performed on up to $N_p = 16$ Intel Delta processors. The measured speedup curve starts to tail off at 16 processors. This is undesirable because it indicates that adding more processors will not result in a proportionate decrease in the wall-clock time. The quantity S_c is also plotted (for reference, for $N_p = 16$, $\tau_{par}/\tau_{CPU} = 0.2$). The difference between S_c and the measured speedup S is due primarily to variation in the on-processor floating point speed as the domains become smaller with increasing N_p. Experiments at $2^o \times 2.5^o$ resolution (not shown) revealed that the speedup curve flattens out above $N_p \approx 20$.

These experiments indicate that a straightforward application of Operator Decomposition, based on a domain-decomposed transport algorithm, would not be effective for the $4^o \times 5^o$ or $2^o \times 2.5^o$ resolutions that are of interest in our work. This is mainly because messages smaller than about one kilobyte (as here) incur a latency (or startup cost) of about 100 μs. One way to avoid this is to concatenate guard-cell data at the beginning of each timestep, and then send the resulting data buffer as a single message. This would add to the complexity of the software. A more serious drawback to the Operator Decomposition is the well-known difficulty of parallelizing the semi-Lagrangian algorithm (e.g., Barros et al. 1995).

An advantage for the Covariance Decomposition is that it is unnecessary to parallelize the transport operator; the choice of transport scheme can be based on scientific merit alone because M is simply implemented as serial code on each processor. The potential disadvantage is that a parallel matrix transpose $(MP)^T$ needs to be implemented. The transpose involves the transfer of almost all the memory of MP (except for diagonal blocks) between processors. This involves more communications (in terms of the total number of bytes) than the Operator Decomposition, where only nearest-neighbor processors communicate via guard cells. However, through the communication of large buffers in the matrix transpose, the effect of message latency is reduced. For example, the time for a global transpose for $4^o \times 5^o$ resolution with 512 processors on the Intel Delta is 0.18 seconds. This compares favorably with the prior estimate of the CPU time to calculate $M(MP)^T$ of about 1 second, leading to an acceptable estimated speedup of $S_c = 512/(1.0+0.18) \approx 434$. Detailed timings for the global transpose (including buffering) for all numbers of processors up to 512 are given in Lyster et al. (1997). In section 4, scaling and timing results for the entire Kalman filter using the Covariance Decomposition are presented.

The Covariance Decomposition approach can be applied to any set of dynamical equations that can be represented in the form of Eq. (9). The only restriction is that the implementation of the operator M should fit on a single processor. For nonlinear dynamics,

the tangent linear model operator would be used to evolve the error covariance (Ménard et al. 1995, Daley 1995).

Our sequential method for evaluating $M(MP)^T$ allocates storage for one matrix of size n^2 and message buffers of size n^2; both of these large memory objects need to be distributed among all processors. In the next section we show that, depending on the number of observations p that are assimilated in a timestep, the memory requirements and number of floating point operations involved in the analysis error covariance computation can compete with (and even exceed) that required for evaluating $M(MP)^T$.

3.3 Implementation of the analysis step

The analysis equations are (11), (12), (13), or (14). The gain K is stored as an $n \times p$ matrix. H is a $p \times n$ sparse operator that interpolates bilinearly from analysis gridpoints to observation locations. In practice, only the four interpolation weights per row of H are actually stored. $P^f H^T$ is $n \times p$, while $HP^f H^T + R$ is $p \times p$. The Kalman filter is a sequential algorithm; at each timestep p observations are assimilated. Since typically $p \ll n$, all of the above matrices are small (as is the state w) compared with size-n^2 matrices, P^f and P^a. The present code stores all small matrices ($n \times p$ and $p \times p$) identically on all processors. This considerably simplifies the software and debugging. The only problem occurs when p is sufficiently large that the storage of the $n \times p$ matrices competes with the storage of size-n^2/N_p components of P on each processor. This occurs when the number of observations in a timestep is $p \approx n/N_p$. For example, at $4^o \times 5^o$ resolution on $N_p = 512$ processors, storage of the small matrices competes with the storage of P when $p \approx 6$ observations per timestep. The Cryogenic Limb Array Etalon Spectrometer (CLAES) instrument on board the UARS satellite retrieves a number of trace constituents in the stratosphere using a limb sounding technique. We are assimilating retrievals from this instrument, and others on board UARS, to generate gridded datasets. In one timestep of our Kalman filter (15 minutes) CLAES produces about 14 observations when interpolated onto an isentropic surface. In this case small-matrix storage dominates that of P. For $2^o \times 2.5^o$ resolution ($Nx = 144, Ny = 90$), $p_{max} = 15$, and $N_p = 512$, the compiled code, including the analysis code, on the Intel Delta requires 12 megabytes per processor, just below the user limit of 12.8 megabytes. In this case, storage of P dominates that of the small matrices, since $n/N_p \approx 26$. The Intel Paragon has twice as much user memory, so runs with $N_p = 256$ are possible at this spatial resolution.

The following summarizes the floating point and communication costs of the analysis equations:

3.3.1 Evaluate the Kalman gain K

The algorithm evaluates contractions where possible so that large size-n^2 matrices are not generated unnecessarily. The first such contraction is $P^f H^T$. For bilinear interpolation, the $p \times n$ matrix H has only four non-zero elements along each row. Each column of the $n \times p$ matrix $P^f H^T$ is therefore a linear combination of four columns of P^f. Thus the evaluation of $P^f H^T$ takes $O(np)$ operations shared over all processors. Since P^f is distributed, and we require K to be reproduced identically on all processors, we first calculate partial sums of $P^f H^T$ on each processor and then perform a global sum over all processors to obtain $P^f H^T$. This is a standard operation on SPMD computers; hence these global-sum routines are usually provided as optimized library calls (usually involving tree-code algorithms, *cf.* Foster 1995). The parallel cost of this is $O(np\,log_2 N_p)$ operations shared over all processors, while the parallel communication cost is optimized according to the architecture of the machine.

The matrix $HP^f H^T$ is evaluated as $H(P^f H^T)$, the matrix $P^f H^T$ already exists on all processors. This takes $O(p^2)$ operations and the global combine takes $O(p^2\,log_2 N_p)$ operations, both shared over all processors, with some communication overhead in the global sum. The observation errors are taken to be uncorrelated; hence R is diagonal, the elements being the measurement error variances. The solution of Eq. (12) to obtain K uses an eigenvalue decomposition to evaluate the inverse of symmetric matrices (Press et al. 1989). This approach allows for the deselection of small eigenvalues in the construction of the inverse of the matrix $HP^f H^T + R$, which is poorly conditioned when the observation error variances are small, especially for perfect observations as in the observability test (see section 5.2). This takes $O(p^3)$ floating point operations per processor to obtain $(HP^f H^T + R)^{-1}$. When our algorithm is used with UARS datasets, ill-conditioned matrices are not expected to arise, in which case we will use a more efficient Cholesky decomposition to solve (12). Finally K is evaluated on each processor as $P^f H^T (HP^f H^T + R)^{-1}$ which takes $O(np^2)$ operations per processor.

The floating point cost of evaluating K, $O(np^2)$ operations on each processor, increases relative to that of $M(MP)^T$, which is $O(hn^2/N_p)$ operations per processor (refer to section 3.1), as p or N_p become larger. There is also a memory burden in storing K and $P^f H^T$ on all processors, which becomes comparable to the storage of P when $p \approx n/N_p$.

3.3.2 Evaluate P^a

Consider first the optimal form Eq. (14): $P^a = (I - KH)P^f$. This is evaluated as $P^f - K(HP^f)$. The second term uses K and $HP^f \equiv (P^f H^T)^T$, both of which are stored identically on all processors. The expansion $K(HP^f)$ is performed in parallel by evaluating only those terms that contribute to each processor's domain for the storage of P^a. This takes $O(n^2 p/N_p)$ operations per processor. This increases relative to the cost of calculating $M(MP)^T$ as p becomes larger.

The Joseph form Eq. (13) is evaluated as:
$P^a = (I - KH)(P^f - K(HP^f))^T + KRK^T$.
Once again this is generated from HP^f, K, and R which are all stored identically on all processors. This operation takes $O(n^2 p/N_p)$ operations per processor, however there is a parallel cost involved in the global transpose of the size-n^2 matrix. Since P^f is overwritten by P^a no additional memory is required, *cf.*, section 3.1.2.

3.3.3 Evaluate w^a

This is carried out identically on all processors. The innovation $w^o - Hw^f$ is a p-vector that is evaluated and saved for collection of innovation statistics. The Kalman gain is applied to this vector and the analyzed state w^a evaluated, Eq. (11). The time to evaluate w^a is dominated by the multiplication by the Kalman gain, which takes $O(np)$ operations per processor.

The matrix inversion and the evaluation of w^a are not parallelized. For these two computations, all processors perform exactly the same calculations and K, HP^f, and w^a are stored identically on each processor. The larger calculations in the analysis step are performed as parallel processes.

4 Timings for the Parallel Kalman Filter

The previous section makes it clear that the Covariance Decomposition strategy is preferred for the covariance forecast dynamics, Eq. (10). We discussed a strategy for the analysis step that involves some global communications to evaluate $P^f H^T$, evaluating K and w^a identically on each processor, and parallelizing the equations for P^a, Eqs. (13) or (14). In this section all timings were obtained for runs on the Intel Paragon at Caltech. The interprocessor communication bandwidth of this machine is about 5 times faster, and the on-processor speed (flop/s) is about 1.2 times faster than that of the Delta. We used single precision arithmetic with compiler optimization options $O4$ and *noieee*.

For medium resolution ($4^o \times 5^o$) using the Joseph form, Eq. (13), Figure 3 shows the ideal speedup ($S_{ideal} = N_p$), as well as the measured speedup for the forecast step, the analysis step, and the full Kalman filter, for N_p = (16, 32, 64, 128, 256, 512). For experiments involving the assimilation of CLAES data, the timestep is 15 minutes and the average number of observations (p) per timestep is 14. The results in this section apply to this case. Note that the minimum number of processors on which this problem was run is 16, so these actual speedups are measured with respect to the times on 16 processors. This speedup is slightly more optimistic than the usual value measured with respect to time on a single processor. However, what is important is the change in speedup as more processors are added to a problem, because this indicates how well the incremental processors are utilized.

Figure 3 indicates that the speedup for the analysis step is less linear (scalable) than for the forecast step, thus degrading scalability of the full Kalman filter. Both steps involve substantial interprocessor communication, and the improvement in on-processor speeds with optimization emphasizes the relative cost of the interprocessor communications (the forecast step is less scalable than was estimated in section 3.2). That is, although the code runs faster with more processors, the scaling is poorer; this is a common result of on-processor optimization. The speedup for the analysis step tails off more quickly than that of the forecast because only part of this step is fully parallelized, namely, the evaluation of P^a.

The total speedup curve in Figure 3 begins to flatten above 256 processors, so that using more than 256 processors at medium resolution for the Joseph form with optimized code does not reduce the wall-clock time significantly. Figure 4 shows the corresponding speedup curves when the optimal form, Eq. (14), is used. Here the time to evaluate P^a is reduced relative to that of K and $P^f H^T$. Since the evaluation of P^a is fully parallel, the analysis step speedup curve now falls off more rapidly than in Figure 3. In fact, the analysis step shows little speedup above 128 processors.

The actual times in seconds per timestep for the analysis using the Joseph form, the forecast step, and the full Kalman filter are shown in Figure 5 for medium resolution and $p = 14$ observations per timestep. The dominant cost of the analysis for large numbers of processors is clear. A typical 10-day run takes 960 timesteps. This evaluates to an acceptable 45 minutes of wall-clock time for the full Kalman filter using 256 processors.

The corresponding results for the optimal form are shown in Figure 6. Since the optimal form is simpler (with fewer floating point operations and without the need for the global transpose), the actual times for the analysis are relatively small. This is why the speedup (scaling) for the full Kalman filter is a little better for the optimal form than for the Joseph form (compare Figs. 3 and 4). Only for large numbers of processors $N_p > 256$ does the time for the analysis step exceed that of the forecast step. The full Kalman filter step takes less

time for the optimal form than the Joseph form, for all numbers of processors. A 10-day run for the optimal form takes about 34 minutes of wall-clock time for the full Kalman filter using 256 processors.

Due to the limitations of main memory, high-resolution runs ($2^o \times 2.5^o$) can only be performed on 256 and 512 processors of the Intel Paragon. Therefore complete speedup curves cannot be plotted; however, comparisons with medium-resolution runs can be made. For a 10-day run with 960 timesteps on 512 processors, the total time for the full Kalman filter at high resolution is 7.8 hours for the Joseph form and 5.0 hours for the optimal form. The ratio of the total time for 256 processors to that of 512 processors is 1.50 for the Joseph form and 1.52 for the optimal form. This scaling is considerably better than for medium resolution, due to the improved scaling of the global transpose for larger sized matrices and the reduced relative cost of calculating the matrices K and $P^f H^T$, at least one of whose dimension is fixed (p).

Actual flop/s rates were calculated using the hardware performance monitor (hpm) on the Goddard Cray C98 to measure the number of floating point operations. The flop/s rates were calculated by dividing the hpm numbers by the actual times (Figures 5 and 6, i.e., for $p = 14$) on the Intel Paragon. Figure 7 shows the gigaflop/s rates for the full Kalman filter (optimal form) for both medium ($4^o \times 5^o$) and high ($2^o \times 2.5^o$) resolutions. We obtain a peak performance of about 1.3 gigaflop/s. This is typical for the i860 RISC-based processors, where local memory-to-memory data transfers reduce the actual throughput below the rated peak (especially for a semi-Lagrangian transport algorithm). The gigaflop/s rates for the Joseph form (not shown) are almost the same as for the optimal form, peaking at 1.2 gigaflop/s; the slight reduction arises from the parallel cost of the extra global transpose operation. We note that there are different interpretations of the term flop/s in the evaluation of parallel code performance. We have used the conservative approach of considering only the number of floating point operations for the serial version of the code on the Cray C98. In deriving the numbers for Figure 7 we do not factor in the extra parallel floating point burden associated with, for example, the global sum in calculating $P^f H^T$.

Both forms of the Kalman filter (Joseph and optimal) scale well up to 256 processors at $4^o \times 5^o$ resolution. Scaling is satisfactory up to 512 processors at $2^o \times 2.5^o$ resolution. The algorithms for evaluating $P^f H^T$ and K are the dominant cause of diminishing speedup. Table 1 shows that the percentages of times taken by $P^f H^T$ and K increase significantly from $N_p = 16$ to 512 processors. In the case of $P^f H^T$ recall that global sum operations are used to combine partial sums over processors. For $p = 14$ and $N_p \gg p$ most processors will make no contribution to the sum, yet the global sum is over all processors. This gives rise to the poor scaling for $P^f H^T$. An optimized algorithm that replaced the global sums

would be considerably more complex. The evaluation of K is not parallelized; the inverse of (HP^fH^T+R), a $p\times p$ matrix, is performed identically on all processors and gives rise to the poor scaling in Table 1. No UARS instrument provides enough observations per timestep to make satisfactory use of a parallel inverse, such as from the Scalapack software library.

We have not found other than bitwise identical results for the same run performed on different numbers of processors. However, because of the use of the global sums that may evaluate partial sums in a different order (depending on N_p and the location of observations), bitwise identical results are not guaranteed by our algorithm.

Table 1. Times for the P^fH^T and K steps as a percentage of the total analysis times for $4^o\times 5^o$ resolution, and 14 observations per timestep. These numbers are evaluated for both 16 and 512 processors. The remaining percentages are dominated by the cost of evaluating P^a, which is highly parallelized.

Number of Processors	16	512
Percentage P^fH^T	1.5	30.
Percentage K	3.2	19.

5 Numerical Tests

Here we present the results of two validation tests of the Kalman filter code, using synthetic winds and observations. These tests are basic for the Kalman filter algorithm; further work will use actual wind datasets and UARS observations. We used the transport scheme of Lin and Rood (1996), which is less diffusive than the van Leer scheme. The algorithm was rendered linear with respect to the constituent density by removal of the monotonicity condition.

5.1 Consistent evolution of the error variance

For non-divergent flows, in the absence of observations, the variance $P(\mathbf{x},\mathbf{x},t)$ satisfies the advection equation (Cohn 1993)

$$\frac{\partial}{\partial t}P(\mathbf{x},\mathbf{x},t)+\mathbf{v}_\theta\cdot\nabla P(\mathbf{x},\mathbf{x},t)=0, \tag{16}$$

where $\mathbf{x}$ denotes a point on the isentropic surface $\theta = constant$. The non-divergent flow considered here is solid-body rotation. In this case Eq. (16) implies that the variance field simply rotates along with the flow, and verifying this property constitutes a test of the implementation of the discrete covariance propagation equation (10). The axis of rotation is chosen to pass through the equator (i.e., flow is over the poles) so that, in particular, this provides a test of the variance propagation near the poles.

A case is presented with $8^o \times 10^o$ resolution ($Nx = 36$ and $Ny = 22$). The timestep is set to 15 minutes, so that one day corresponds to 96 timesteps. The rotation period is 1 day. In this case the maximum Courant number for flow at the equator is $44/96 = 0.46$. The initial error covariance function is chosen to have a space-limited cosine structure:

$$P(\mathbf{x_1}, \mathbf{x_2}, t = 0) = \begin{cases} 0.25(1 + cos(\pi\theta_1/\theta_a))(1 + cos(\pi\theta_2/\theta_a)) \\ \qquad for\ 0 \le \theta_1 \le \theta_a\ and\ 0 \le \theta_2 \le \theta_a \\ 0. \qquad for\ \theta_1 > \theta_a\ or\ \theta_2 > \theta_a \end{cases} \tag{17}$$

where $\theta_1 = \theta(\mathbf{x_1})$, $\theta_2 = \theta(\mathbf{x_2})$, and $\theta(\mathbf{x})$ is the great-circle angle between $\mathbf{x}$ and a fixed point on the equator where the solid-body speed is a maximum. The initial variance $P(\mathbf{x}, \mathbf{x}, t = 0)$ is therefore a squared cosine hill centered at the equator. Since $P(\mathbf{x_1}, \mathbf{x_2}, t = 0)$ given by Eq. (17) is a product $f(\mathbf{x_1})f(\mathbf{x_2})$ with f continuous, it follows that $P(\mathbf{x_1}, \mathbf{x_2}, t = 0)$ is a legitimate covariance function (Gaspari and Cohn 1996). The initial covariance matrix P_0^a is obtained by evaluating Eq. (17) on the grid.

Figure 8(a) shows a contour plot of the initial variance field evaluated on the $8^o \times 10^o$ grid. For this case $\theta_a = 21\pi/64$, so the total width of the structure is about 120^o (i.e., 12 grid points in longitude and 15 in latitude). Figure 8(b) shows the discrete variance field, or diagonal of P, after integrating Eq. (10) for 96 timesteps. Except for a slight north-south asymmetry, the overall shape is well-preserved after the passage over the poles.

The total variance is defined to be the integral

$$V = \int d\mathbf{x}\, P(\mathbf{x}, \mathbf{x}), \tag{18}$$

where $d\mathbf{x}$ is area measured on the surface of the sphere. The integral is evaluated numerically on the grid. For the present case the initial total variance is 0.5589 and the final total variance is 0.5493. The discrete dynamics results in a mild diffusion in the transport of variance over the poles.

5.2 Observability test

The second test involves both forecast and analysis steps, using synthetic perfect observations. The total variance V, as defined in Eq. (18) should reduce to zero (to machine precision) in finite time if the observability condition is met (Cohn and Dee 1988). Solid-body rotation winds are used again, but now with the axis of rotation is through the poles, and again at $8^o \times 10^o$ resolution. The wind rotation period is again one day, but a timestep of 40 minutes is chosen so that the Courant number is everywhere equal to one (the flow is zonal). Observations are made at all grid points along a fixed meridian at each timestep, and the observation error covariance matrix R is taken to be zero. Thus the entire flow is

observed perfectly in one day, so that the observability condition is met and therefore the total variance must reduce to zero in one day. The Joseph formula, Eq. (13), is used to help ensure numerical stability in this extreme case.

The initial error covariance is taken to be the isotropic second-order autoregressive (SOAR) model

$$P(\mathbf{x_1}, \mathbf{x_2}, t = 0) = (1 + (2r_e/L)sin(\theta/2))exp(-(2r_e/L)sin(\theta/2)), \tag{19}$$

where $\theta = \theta(\mathbf{x_1}, \mathbf{x_2})$ is the great-circle angle between positions $\mathbf{x_1}$ and $\mathbf{x_2}$ on the sphere (Weber and Talkner 1993), r_e is the radius of the earth, and L is the correlation length. Figure 9 shows the total variance V (in normalized units of r_e^2) as a function of time for values of correlation length $L = (1{,}000$ km, 500 km, 5 km). The variance is plotted through points taken every 4 timesteps. The initial value of V is 4π since $P(\mathbf{x}, \mathbf{x}, t = 0) = 1$. For the cases $L = 1{,}000$ km and $L = 500$ km, where the correlation length is comparable to the grid spacing near the equator and greatly exceeds the grid spacing near the poles, the variance decreases rapidly at first, then decreases linearly, and finally reaches zero in one day. The case where the correlation length is 5 km is well below the grid spacing, corresponding to an initial covariance structure that is unity on the diagonal of P and small elsewhere. In this case we expect the total variance to decrease almost linearly because from the first timestep there is negligible correlation between nearby gridpoints. This behavior is demonstrated in Figure 9.

6 Summary and Conclusions

We have implemented on distributed-memory parallel computers a Kalman filter for the assimilation of atmospheric constituents on isentropic surfaces over the globe. The code runs at resolutions of $8^o \times 10^o$, $4^o \times 5^o$, and $2^o \times 2.5^o$ on the 512-processor Intel Paragon and Delta machines at the California Institute of Technology, using Fortran 77 with the NX message-passing library. We have developed a Covariance Decomposition approach as the basis for the parallel algorithm. This approach distributes the columns of the forecast/analysis error covariance matrix on different processors. A considerable advantage of this scheme is that it is not necessary to parallelize the model transport code; only that it fits onto the memory of each processor. This approach is also efficient in terms of the distribution of floating point operations and memory, with some parallel cost involved in a global matrix transpose. Ten-day runs using UARS-CLAES observation datasets can be completed in 34 minutes for the optimal form of the analysis at medium resolution ($4^o \times 5^o$) on 256 processors of the Paragon with $O4$ and *noieee* compiler optimizations (45 minutes for the Joseph form). The corresponding high-resolution ($2^o \times 2.5^o$) runs take 5 hours on 512 processors (7.8 hours for

the Joseph form).

The Kalman filter forecast step shows some reduction in scaling when the full 512 processors of the machines are used with compiler optimizations. This reduction is due primarily to communication overhead involved in the global matrix transpose. The reduction in scaling for the Kalman filter analysis step is more severe. This reduction is due primarily to the serial (unparallelized) calculation of the Kalman gain matrix on each processor – sometimes referred to as an Amdahl's bottleneck – and, more significantly, to software simplifications that involve the use of global sum library subroutines.

Overall the peak performance obtained for high-resolution runs on 512 processors of the Paragon is about 1.3 gigaflop/s. This may be improved by on-processor memory-to-memory optimization or evaluating the matrix $P^f H^T$ more directly, using fewer floating point operations and communication calls than do the global sums. We expect to port our code to machines such as the Cray T3E without much effort, improving further the wall-clock time for high-resolution runs.

Basic tests of the parallel Kalman filter code using synthetic data examined variance transport and verified observability properties. The code is now being used to assimilate retrieved constituent data from UARS instruments, using analyzed wind fields from the DAO global atmospheric data assimilation system to drive the transport model. Work on characterizing transport model errors is in progress. Results of these data assimilation studies will be reported in a future publication.

Acknowledgments

PML would like to thank Robert Ferraro of Jet Propulsion Laboratory for his help on the parallel algorithms. RM would like to thank the Canadian Atmospheric Environment Service for its support. This research was performed in part using the CSCC parallel computing system operated by Caltech on behalf of the Concurrent Supercomputing Consortium, and also the NASA Center for Computational Sciences (NCCS) at Goddard Space Flight Center. Access to these facilities as well as support for PML was provided by the NASA High Performance Computing and Communications (HPCC) program Earth and Space Sciences (ESS) project.

7 Appendix

Appendix A. A load balanced Covariance Decomposition

The covariance matrix is indexed $P(i1, j1, i2, j2)$ where $(i1, j1)$ and $(i2, j2)$ are Fortran indices for two positions on a discretized latitude-longitude grid. Following the convention that is used for the state vector w, the entire matrix is dimensioned $P(1:Nx, 0:Ny, 1:Nx, 0:Ny)$. The Covariance Decomposition assigns contiguous columns of P onto different processors in such a way that the totality of all columns on all processors makes up the entire matrix without redundancy. This amounts to a domain decomposition where a range of $(i2, j2)$ is assigned to a processor corresponding to a contiguous sequence on a grid whose Fortran dimension statement has the range $(1{:}Nx, 0{:}Ny)$. Each processor allocates its domain of the matrix as $P(1:Nx, 0:Ny, ib:ie, jb:je)$ where (ib, ie, jb, je) depend on the processor identification number which, by convention, ranges from 0 to $Np-1$. Two situations arise. For the case $Np < Ny+1$ at least one processor must have a range of $j2$ such that $je > jb$; therefore, $ib = 1$ and $ie = Nx$. For the case $Np \geq Ny+1$ it is not necessary that any processor overlap multiple values of $j2$, i.e., $je = jb$. In fact, this condition is necessary to conserve memory when Np is much greater than $Ny + 1$, because it is the only way to impose a limited range on $i2$, i.e., $(ib{:}ie)$ must encompass a range that is less than $(1{:}Nx)$. The load imbalance of the resulting decomposition arises from the uneven numbers of columns of P on different processors. If we define the load imbalance L as the maximum number of columns on a processor divided by the minimum number, then it can be shown (Lyster et al. 1997) that the worst case occurs when $Np = Ny + 1$, corresponding to $L_{max} = (Nx + 1)/Nx$. For all other cases L is closer to unity. Clearly, for problems of interest (e.g., for $4^o \times 5^o$ resolution $Nx = 72$) load imbalance is not a problem.

References

Allen, D. J., A. R. Douglass, R. B. Rood, and P. D. Guthrie, 1991: Application of a monotonic upstream-biased transport scheme to three-dimensional constituent transport calculations. *Mon. Wea. Rev.*, **119**, 2456-2464.

Andrews, D. G., J. R. Holton, and C. B. Leovy, 1987: *Middle Atmosphere Dynamics*. Academic Press, New York, 489 pp.

Barros, S. R. M, D. Dent, L. Isaksen, and G. Robinson, 1995: The IFS model: overview and parallel strategies, *Coming of Age: proceedings of the sixth ECMWF workshop on the use of parallel processors in meteorology*. Eds. G-R. Hoffmann and K Kreitz, World Scientific, 303-318, ISBN 981-02-2211-4.

Brasseur, G., and S. Solomon, 1984: *Aeronomy of the Middle Atmosphere*. Reidel, Dordrecht, Netherlands, 441 pp.

Bucy, R. S, and P. D. Joseph, 1968: *Filtering for Stochastic Processes with Applications to Guidance*. Wiley-Interscience, 195 pp.

Cohn, S. E., 1993: Dynamics of short-term univariate forecast error covariances. *Mon. Wea. Rev.*, **121**, 3123-3149.

———, 1997: An introduction to estimation theory. NASA/Goddard Space Flight Center Data Assimilation Office Note 97-01, 75pp. *J. Meteor. Soc. Japan*, accepted. Available from
`http://dao.gsfc.nasa.gov/subpages/office-notes.html`

———, and D. P. Dee, 1988: Observability of discretized partial differential equations. *SIAM J. Numer. Anal.*, **25**, 586-617.

———, and D. F. Parrish, 1991: The behavior of forecast error covariances for a Kalman filter in two dimensions, *Mon. Wea. Rev.*, **119**, 1757-1785.

———, and R. Todling, 1996: Approximate data assimilation schemes for stable and unstable dynamics. *J. Met. Soc. Japan*, **74**, 63-75.

Daley R., 1995: Estimating the wind field from chemical constituent observations: Experiments with a one-dimensional extended Kalman filter. *Mon. Wea. Rev.*, **123**, 181-198.

da Silva, A. M., J. Pfaendtner, J. Guo, M. Sienkiewicz, and S. E. Cohn, 1995: Assessing the effects of data selection with DAO's Physical-space Statistical Analysis System. *Second International Symposium on Assimilation of*

Observations in Meteorology and Oceanography, Tokyo, 13-17 March 1995, World Meteorological Organization, pp 273-278.

Dee, D. P., 1995: On-line estimation of error covariance parameters for atmospheric data assimilation. *Mon. Wea. Rev.*, **123**, 1128-1145.

Foster, I. T., 1995: *Designing and Building Parallel Programs: Concepts and Tools for Parallel Software Engineering*, Addison-Wesley, 381 pp.

Gaspari, G. and S. E. Cohn, 1996: Construction of Correlation Functions in Two and Three Dimensions. NASA/Goddard Space Flight Center Data Assimilation Office Note 96-03. Available from

`http://dao.gsfc.nasa.gov/subpages/office-notes.html`

Gelb, A., (ed.), 1974: *Applied Optimal Estimation.* M.I.T. Press, Cambridge, Massachusetts, 374 pp.

Ghil, M., S. E. Cohn, J. Tavantzis, K. Bube, and E. Isaacson, 1981: Applications of estimation theory to numerical weather prediction. *Dynamic Meteorology: Data Assimilation Methods*, L. Bengtsson, M. Ghil, and E. Källen, Eds. Springer-Verlag, 330 pp.

Jazwinski, A. H., 1970: *Stochastic Processes and Filtering Theory.* Academic Press, 276pp.

Lin, S.-J., and R. B. Rood, 1996: Multidimensional Flux-Form Semi-Lagrangian Transport Schemes. *Mon. Wea. Rev.*, **124**, 2046-2070.

Lyster, P. M., S. E. Cohn, R. Ménard, and L.-P. Chang, 1997: A Domain Decomposition for Covariance Matrices based on a Latitude-Longitude Grid, NASA/Goddard Space Flight Center Data Assimilation Office Note 97-03, 35pp. Available from

`http://dao.gsfc.nasa.gov/subpages/office-notes.html`

Ménard, R., 1994: *Kalman filtering of Burger's equation and its application to atmospheric data assimilation.* Ph.D. Thesis, Mc Gill University, Canada. Stormy Weather Group scientific report NW-100, 211p. [Available from the Department of Atmospheric and Oceanic Sciences, Mc Gill University, 805 Sherbrooke Street West, Montreal, Quebec, H3A 2K6, Canada and also from University Microfiche, 300 North Zeeb Road, P.O. Box 1346, Ann Arbor, Michigan, 8106-1346].

———, P. M. Lyster, L.-P. Chang, and S. E. Cohn, 1995: Middle atmosphere assimilation of UARS constituent data using Kalman filtering: preliminary

results. *Second International Symposium on Assimilation of Observations in Meteorology and Oceanography*, Tokyo, 13-17 March 1995, World Meteorological Organization, pp 235-238.

Press, W. H., B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, 1989: *Numerical Recipes: the art of scientific computing.* Cambridge University Press, New York, 818 pp.

Reber, C. A. 1993: The Upper Atmosphere Research Satellite (UARS). *Geophys. Res. Lett.*, **20**, 1215-1218.

Riishøjgaard, L. P., 1996: On four-dimensional variational assimilation of ozone data in weather-prediction models. *Q. J. Roy. Meteorol. Soc.*, **122**, 1545-1571.

Rood, R. B. 1987: Numerical advection algorithms and their role in atmospheric transport and chemistry models. *Rev. Geophys.*, **25**, 71-100.

———, and M. A. Geller, 1994: The Upper Atmosphere Research Satellite: Early Scientific Results. *J. Atmos. Sci.*, **51**, 20, 2783-3105.

———, A. R. Douglass, J. A. Kaye, M. A. Geller, C. Yuechen, D. J. Allen, E. M. Larson, E. R. Nash, and J. E. Nielsen, 1991: Three-Dimensional Simulations of Wintertime Ozone Variability in the Lower Stratosphere. *J. Geophys. Res.*, **96**, 5055-5071.

Todling, R., and S. E. Cohn, 1994: Suboptimal schemes for atmospheric data assimilation based on the Kalman filter. *Mon. Wea. Rev.*, **122**, 2530-2557.

Weber, R. O., and P. Talkner, 1993: Some Remarks on Spatial Correlation Function Models. *Mon. Wea. Rev.*, **121**, 2611-2617.

Williamson, D. L., and P. J. Rasch, 1989: Two-dimensional semi-Lagrangian transport with shape preserving interpolation. *Mon. Wea. Rev.*, **117**, 102-129.

Table 1. Times for the $P^f H^T$ and K steps as a percentage of the total analysis times for $4^o \times 5^o$ resolution, and 14 observations per timestep. These numbers are evaluated for both 16 and 512 processors. The remaining percentages are dominated by the cost of evaluating P^a, which is highly parallelized.

Number of Processors	16	512
Percentage $P^f H^T$	1.5	30.
Percentage K	3.2	19.

Figure captions

Figure 1 (a). Schematic for the *Operator Decomposition* approach for storing large size-n^2 matrices and performing $M(MP)^T$.

Figure 1 (b). Schematic for the *Covariance Decomposition* approach for storing large size-n^2 matrices and performing $M(MP)^T$.

Figure 2. Speedup curves for the domain decomposed van Leer transport algorithm implemented on the Intel Delta.

Figure 3. The actual speedups for the forecast step, the analysis step, and the full Kalman filter on the Intel Paragon for medium resolution ($4^o \times 5^o$) using Covariance Decomposition and the Joseph form Eq. (13).

Figure 4. The actual speedups for the forecast step, the analysis step, and the full Kalman filter on the Intel Paragon for medium resolution ($4^o \times 5^o$) using Covariance Decomposition and the optimal form Eq. (14).

Figure 5. The actual time (seconds) per timestep of the forecast step, the analysis step, and the full Kalman filter on the Intel Paragon for medium resolution ($4^o \times 5^o$) using Covariance Decomposition and the Joseph form Eq. (13).

Figure 6. The actual time (seconds) per timestep of the forecast step, the analysis step, and the full Kalman filter on the Intel Paragon for medium resolution ($4^o \times 5^o$) using Covariance Decomposition and the optimal form Eq. (14).

Figure 7. Gigaflop/s rates for the full Kalman filter at medium and high resolution on the Intel Paragon. These numbers were obtained for the optimal form of the analysis step, Eq. (14).

Figure 8. For solid body wind propagation over the poles: (a) the initial variance; (b) the final variance after a full rotation period of the winds.

Figure 9. The total variance V versus time for a meridional observing network, and an observation error covariance matrix $R = 0$. The initial error covariance matrix is obtained from the SOAR covariance function with values of correlation length $L = (1{,}000$ km, 500 km, 5 km). The rotation period of the solid body winds about the polar axis is one day.

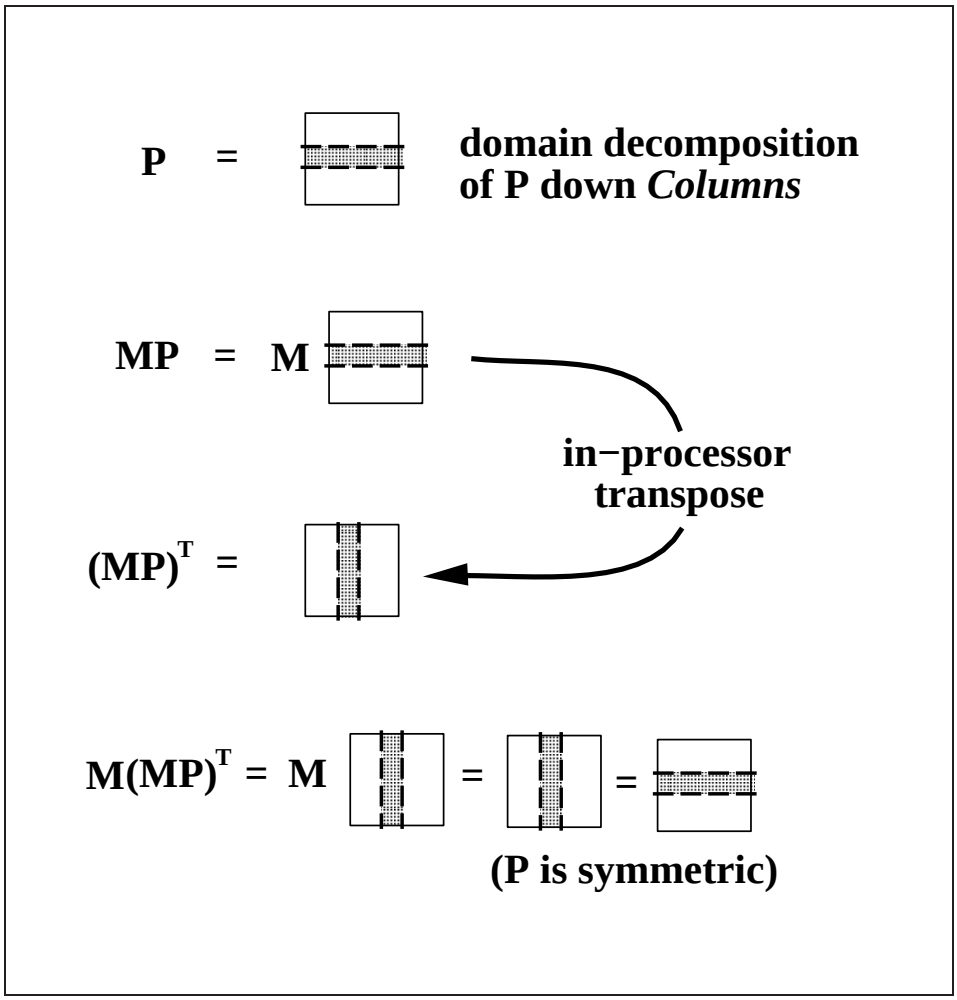

Figure 1 (a). Schematic for the *Operator Decomposition* approach for storing large size-n^2 matrices and performing $M(MP)^T$.

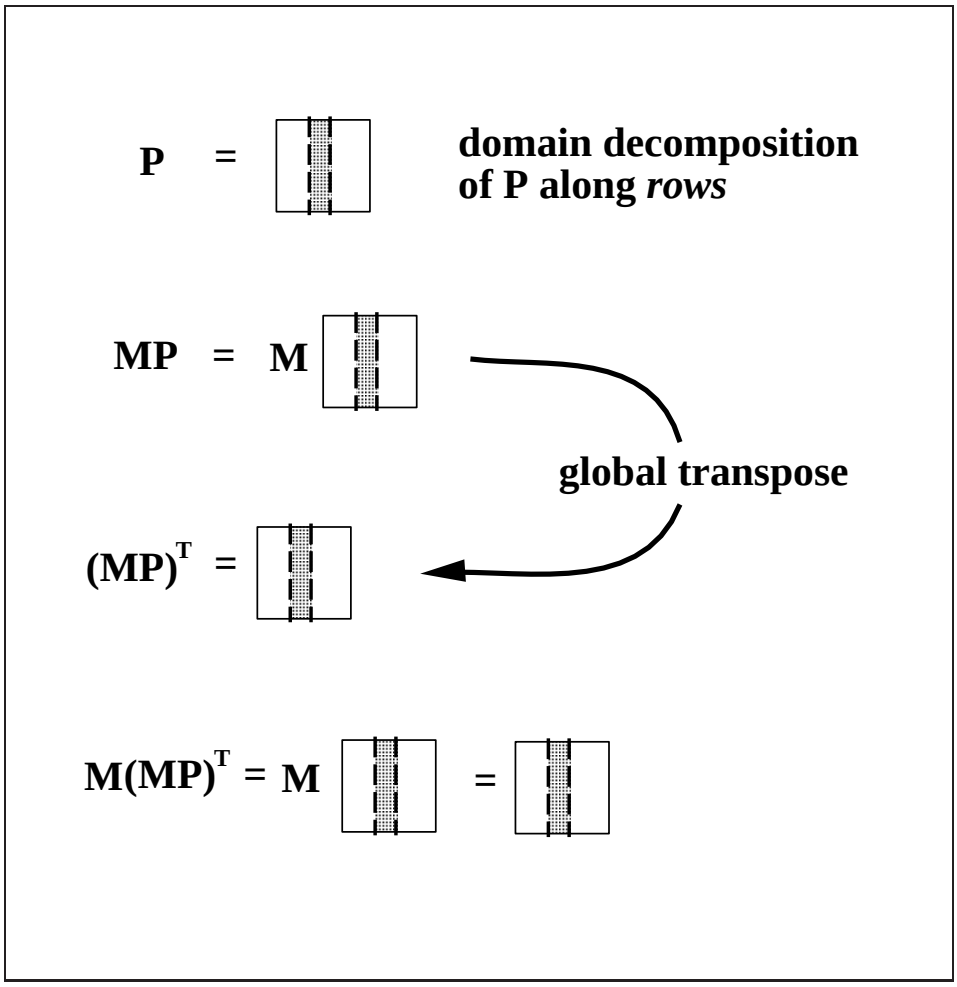

Figure 1 (b). Schematic for the *Covariance Decomposition* approach for storing large size-n^2 matrices and performing $M(MP)^T$.

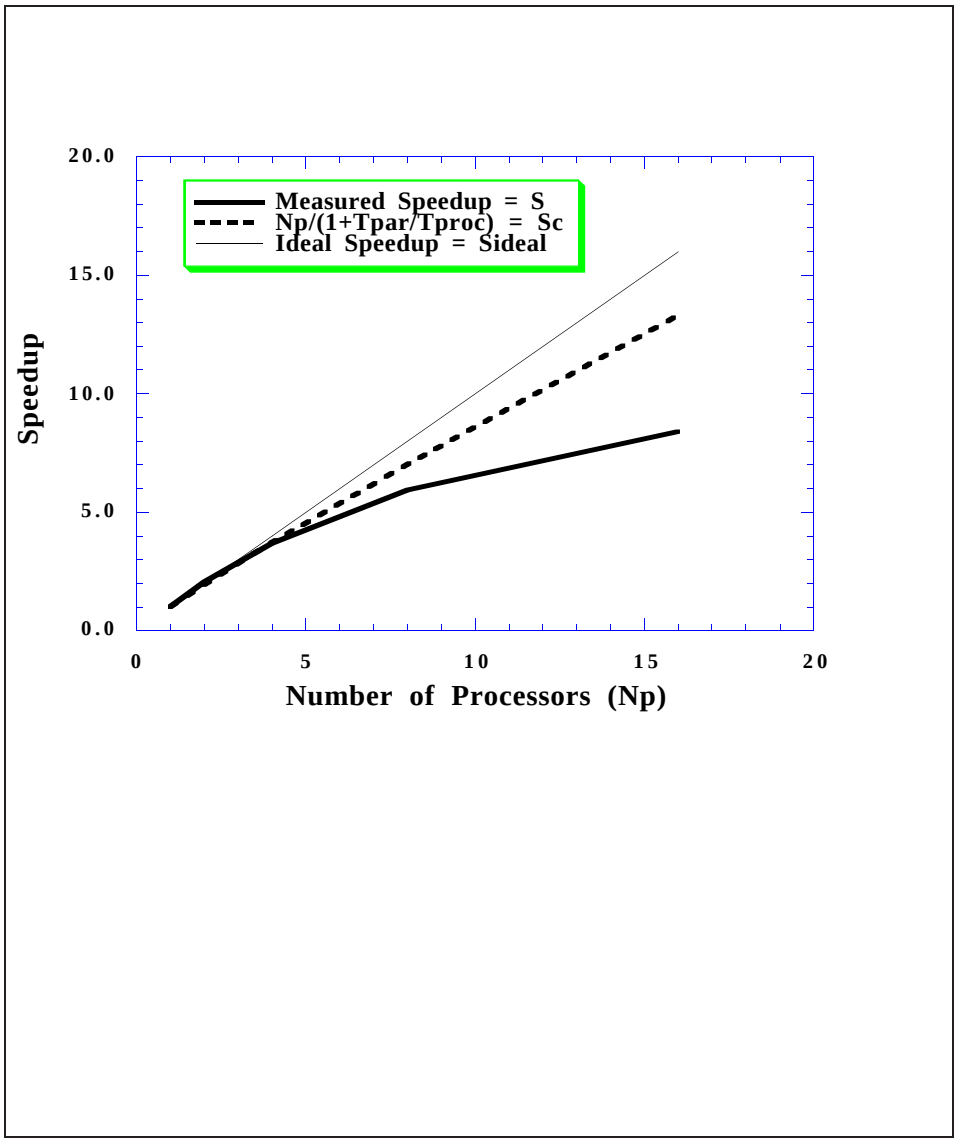

Figure 2. Speedup curves for the domain decomposed van Leer transport algorithm implemented on the Intel Delta.

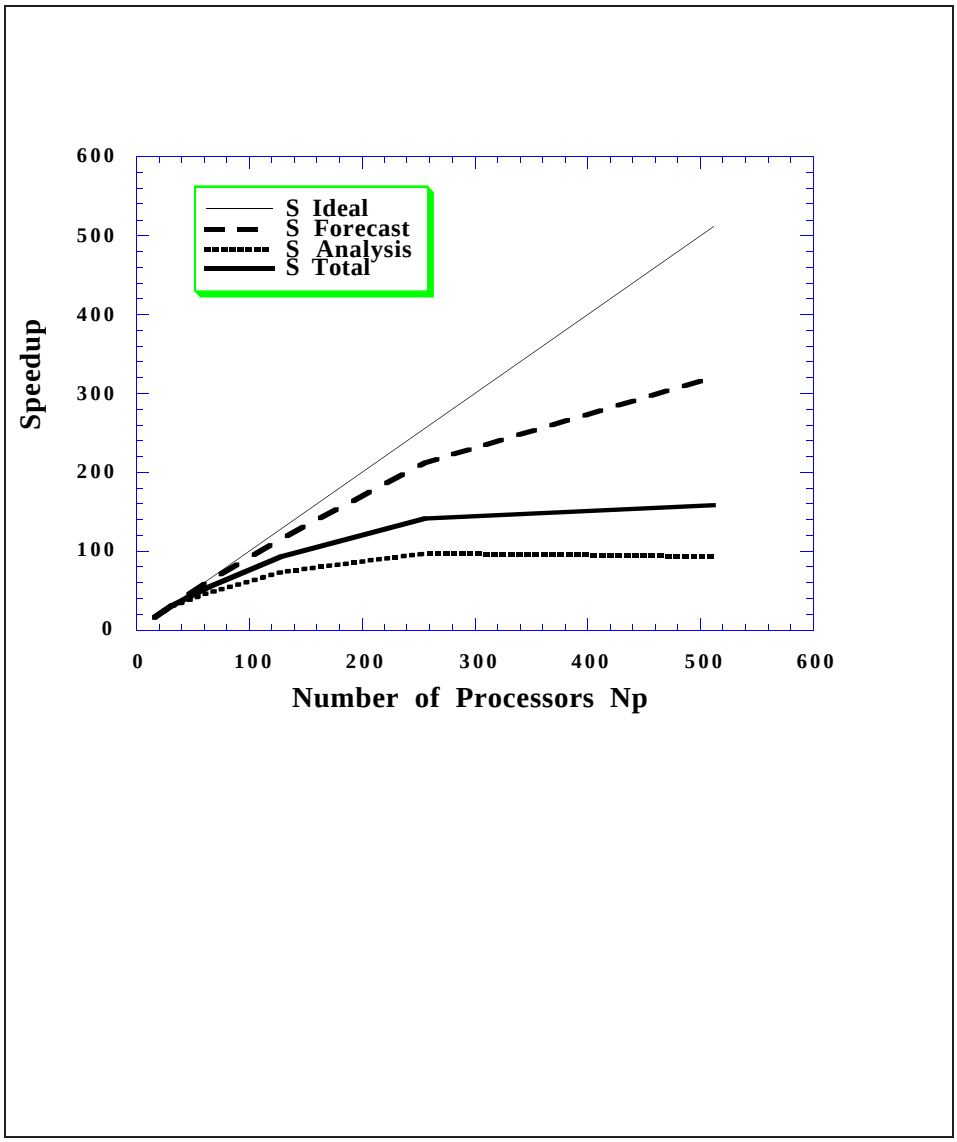

Figure 3. The actual speedups for the forecast step, the analysis step, and the full Kalman filter on the Intel Paragon for medium resolution ($4^o \times 5^o$) using Covariance Decomposition and the Joseph form Eq. (13).

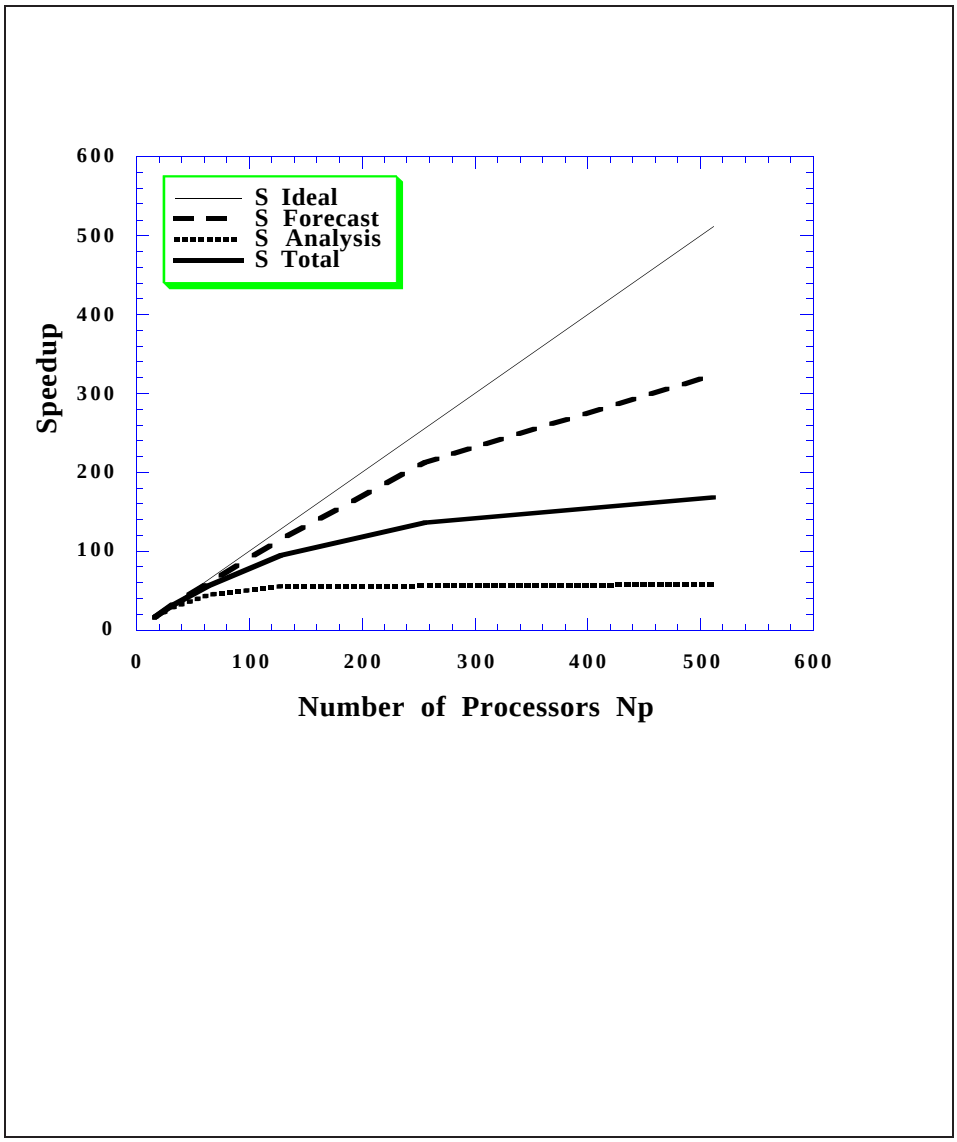

Figure 4. The actual speedups for the forecast step, the analysis step, and the full Kalman filter on the Intel Paragon for medium resolution ($4^o \times 5^o$) using Covariance Decomposition and the optimal form Eq. (14).

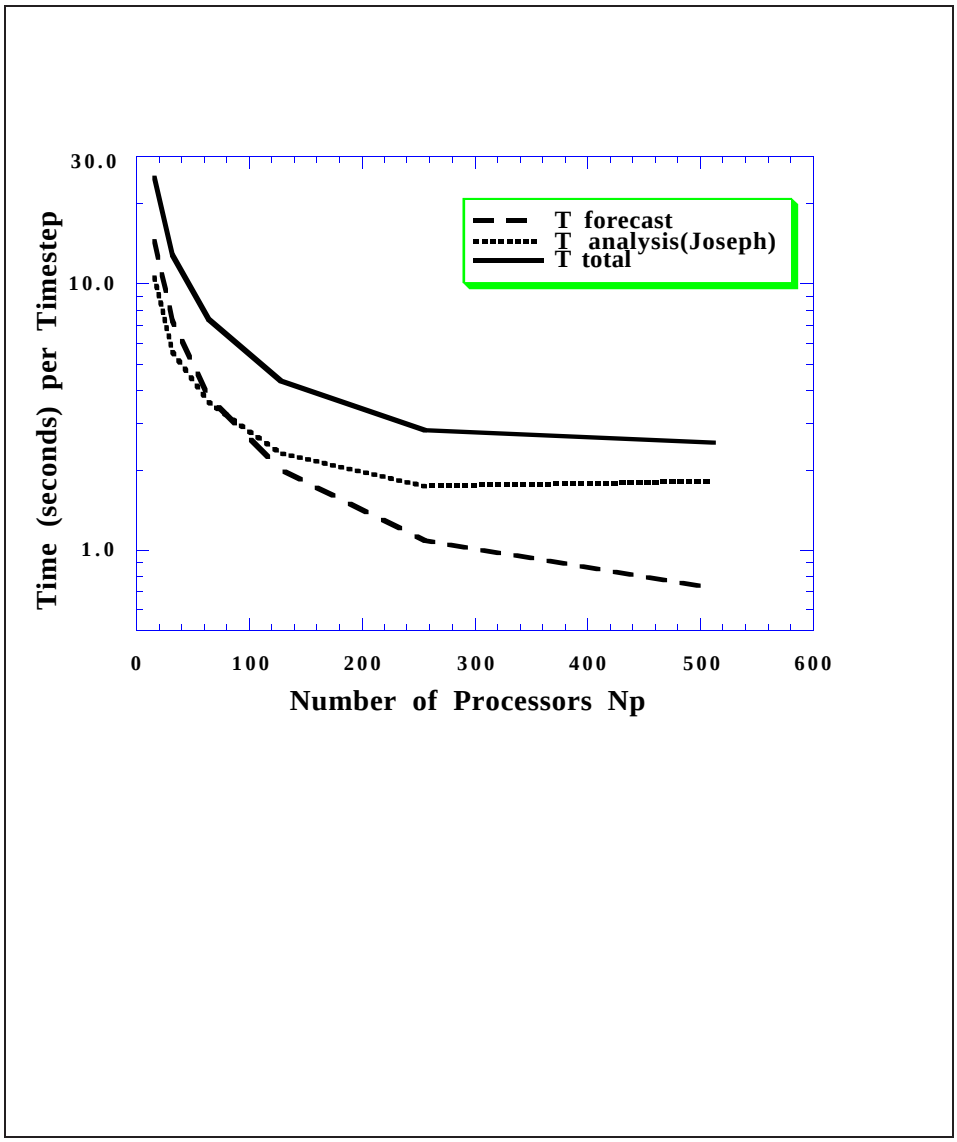

Figure 5. The actual time (seconds) per timestep of the forecast step, the analysis step, and the full Kalman filter on the Intel Paragon for medium resolution ($4^o \times 5^o$) using Covariance Decomposition and the Joseph form Eq. (13).

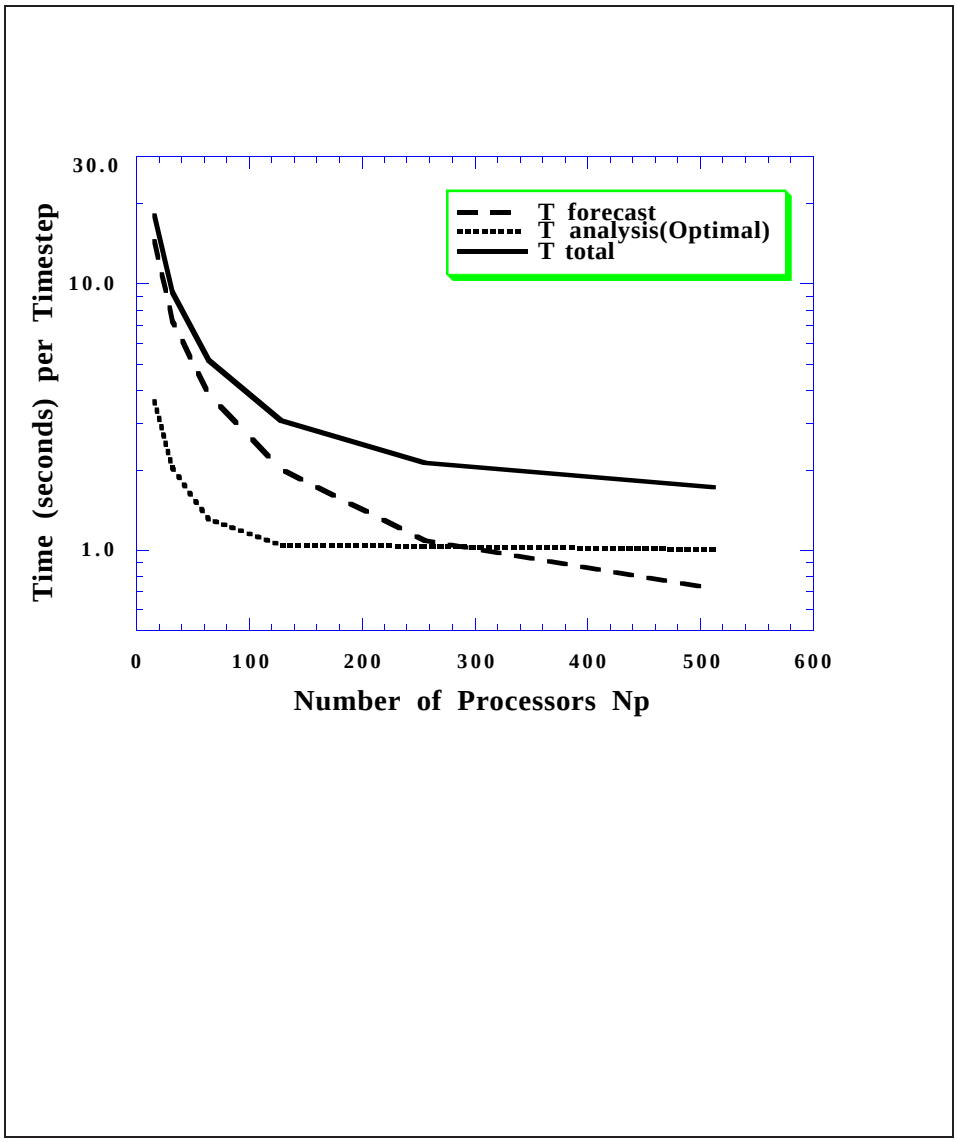

Figure 6. The actual time (seconds) per timestep of the forecast step, the analysis step, and the full Kalman filter on the Intel Paragon for medium resolution ($4^o \times 5^o$) using Covariance Decomposition and the optimal form Eq. (14).

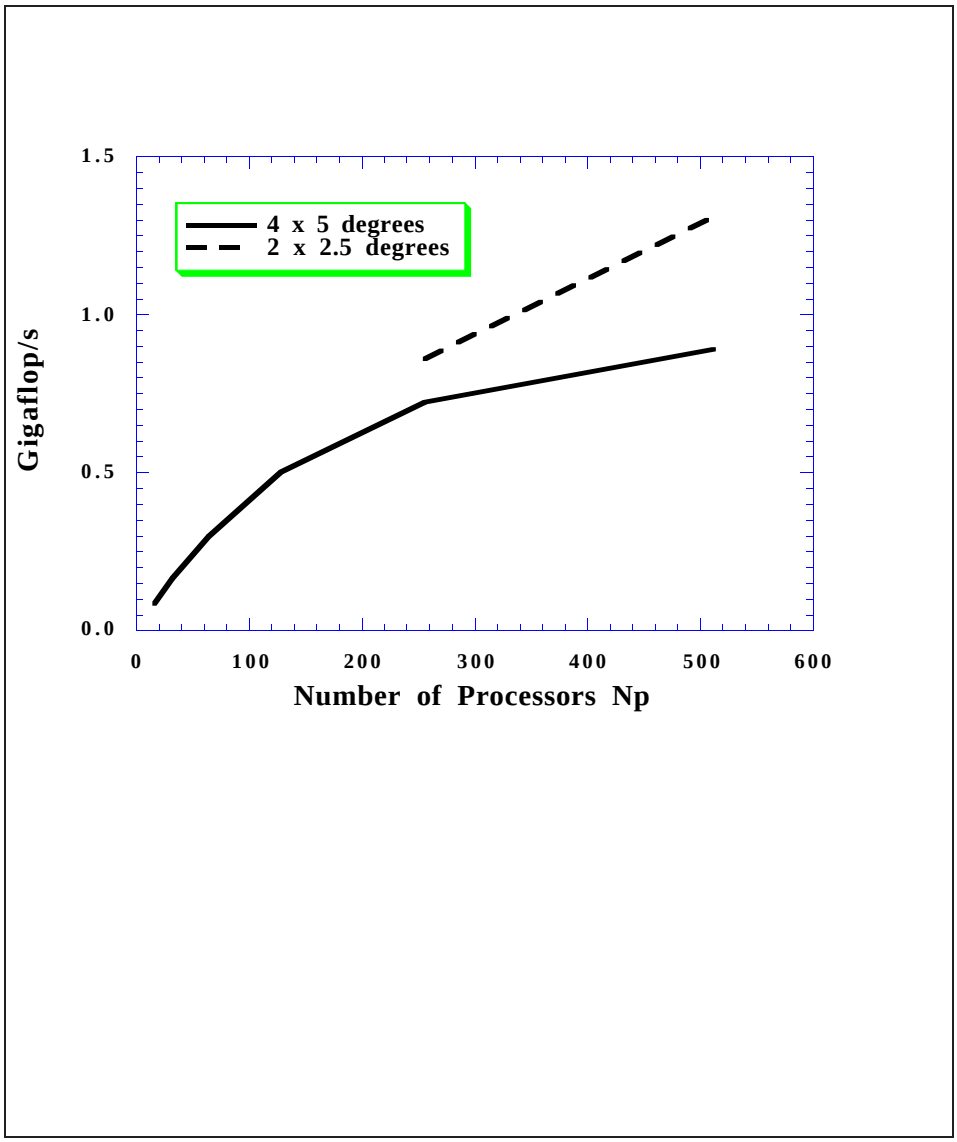

Figure 7. Gigaflop/s rates for the full Kalman filter at medium and high resolution on the Intel Paragon. These numbers were obtained for the optimal form of the analysis step, Eq. (14).

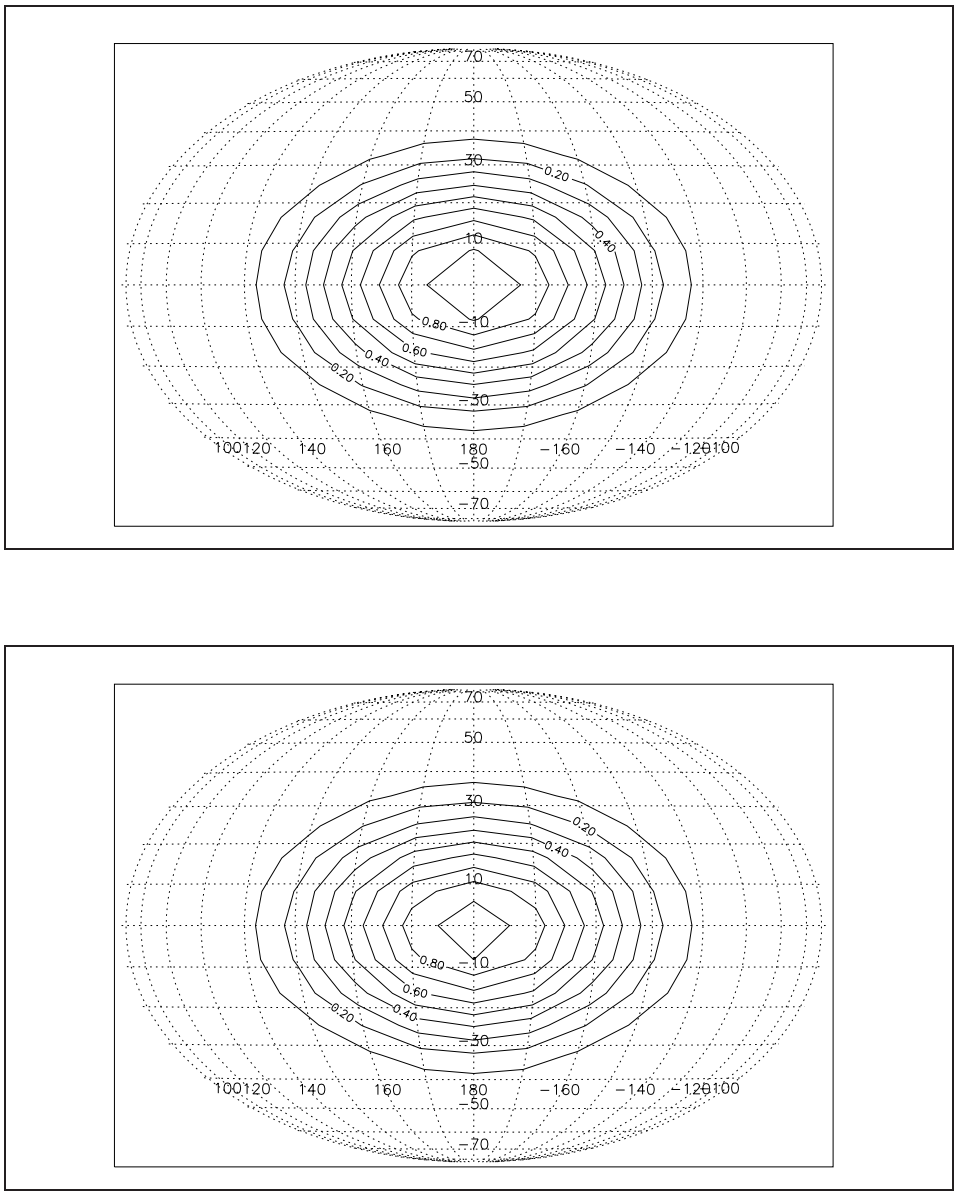

Figure 8. For solid body wind propagation over the poles: (a) the initial variance; (b) the final variance after a full rotation period of the winds.

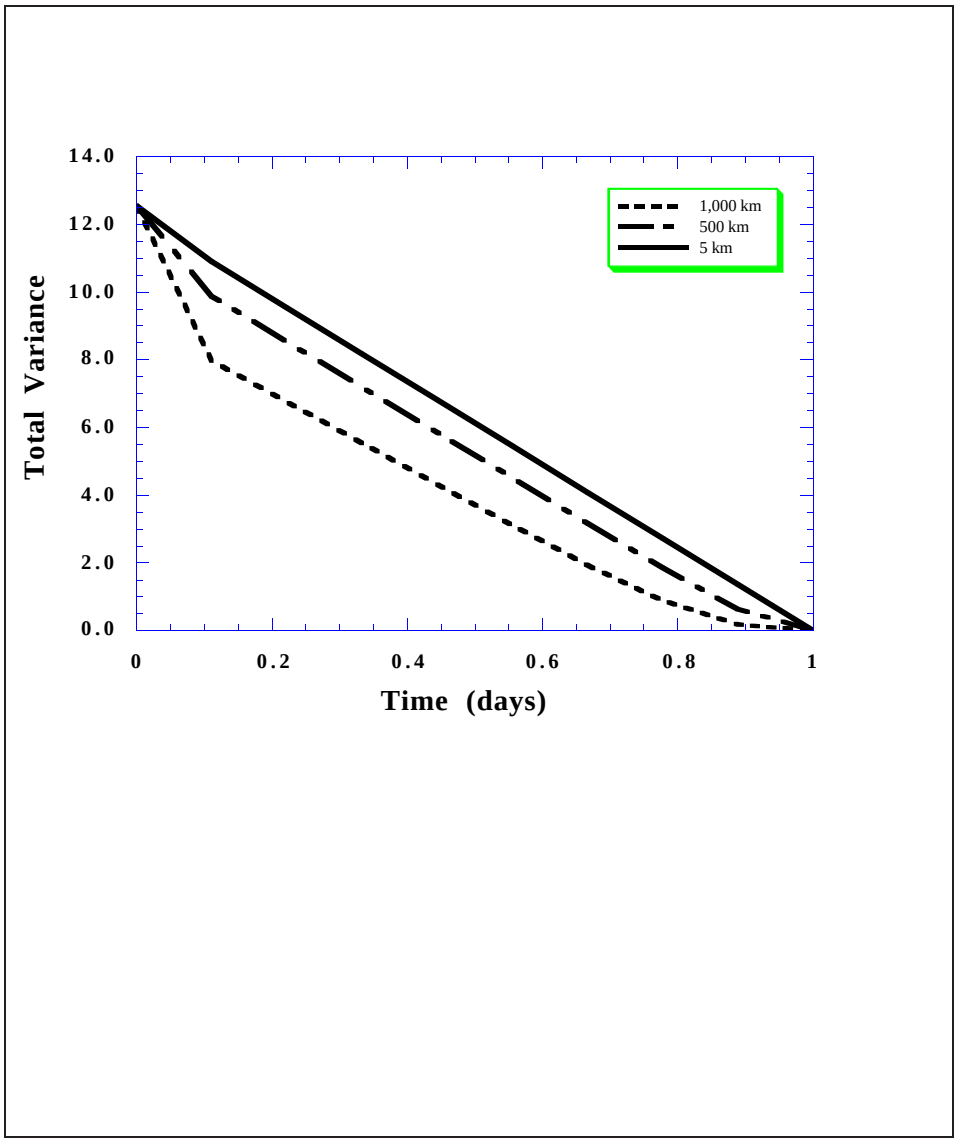

Figure 9. The total variance V versus time for a meridional observing network, and an observation error covariance matrix $R = 0$. The initial error covariance matrix is obtained from the SOAR covariance function with values of correlation length $L = (1{,}000$ km, 500 km, 5 km). The rotation period of the solid body winds about the polar axis is one day.