Scalable Implementations of Ensemble Filter Algorithms for Data Assimilation

JEFFREY L. ANDERSON AND NANCY COLLINS

Data Assimilation Research Section, National Center for Atmospheric Research, Boulder, Colorado

(Manuscript received 10 April 2006, in final form 17 November 2006)

ABSTRACT

A variant of a least squares ensemble (Kalman) filter that is suitable for implementation on parallel architectures is presented. This parallel ensemble filter produces results that are identical to those from sequential algorithms already described in the literature when forward observation operators that relate the model state vector to the expected value of observations are linear (although actual results may differ due to floating point arithmetic round-off error). For nonlinear forward observation operators, the sequential and parallel algorithms solve different linear approximations to the full problem but produce qualitatively similar results. The parallel algorithm can be implemented to produce identical answers with the state variable prior ensembles arbitrarily partitioned onto a set of processors for the assimilation step (no caveat on round-off is needed for this result).

Example implementations of the parallel algorithm are described for environments with low (high) communication latency and cost. Hybrids of these implementations and the traditional sequential ensemble filter can be designed to optimize performance for a variety of parallel computing environments. For large models on machines with good communications, it is possible to implement the parallel algorithm to scale efficiently to thousands of processors while bit-wise reproducing the results from a single processor implementation. Timing results on several Linux clusters are presented from an implementation appropriate for machines with low-latency communication.

Most ensemble Kalman filter variants that have appeared in the literature differ only in the details of how a prior ensemble estimate of a scalar observation is updated given an observed value and the observational error distribution. These details do not impact other parts of either the sequential or parallel filter algorithms here, so a variety of ensemble filters including ensemble square root and perturbed observations filters can be used with all the implementations described.

1. Introduction

Ensemble (Kalman) filters are becoming increasingly popular tools for doing data assimilation in ocean and atmospheric applications. These methods can be derived as ensemble extensions to classical Kalman filters (Kalman 1960; Evensen 1994; Houtekamer and Mitchell 1998; Whitaker and Hamill 2002) or as Monte Carlo approximations to the Bayesian filtering problem (Jazwinski 1970; Anderson and Anderson 1999). Anderson (2003, hereafter A03) presented a sequential least squares framework for implementing most variants of ensemble filters that have been described in the literature. In this framework, it is possible to completely describe an ensemble filter by discussing only the impact of a single scalar observation on a single state variable. Several filters have been implemented using this framework in both idealized and large models (Zhang et al. 2005). However, the sequential nature of the algorithm has led to concerns that it cannot be practically implemented for large problems on massively parallel computers. Several other methods for parallelizing filters have been presented (Ott et al. 2004; Keppenne and Rienecker 2002; Houtekamer and Mitchell 2001). Here, a parallel algorithm is developed in the least squares framework. When forward observation operators are linear, the sequential algorithm in A03 and the parallel algorithm are proved to be identical although actual results may differ due to round-off error from finite precision arithmetic. An important feature of the parallel algorithm is that prior ensembles of state variables can be partitioned onto any number of processors in an arbitrary fashion for assimilation and produce identical answers (even with finite precision) to those from a single-processor implementation. This is particularly useful when developing filters for large

Corresponding author address: Jeffrey L. Anderson, NCAR, P.O. Box 3000, Boulder, CO 80307-3000.
E-mail: jla@ucar.edu

DOI: 10.1175/JTECH2049.1

models with sensitive dependence on initial conditions where bit-wise differences can lead to large differences later in an assimilation.

The simplicity of the least squares framework is maintained, making embellishments such as quality control, sampling error correction and localization (Houtekamer and Mitchell 2001; Mitchell et al. 2002), and covariance inflation (Anderson and Anderson 1999) straightforward to implement in the parallel algorithm. The flexibility in partitioning prior state variable ensembles allows implementations that are efficient on a variety of parallel architectures. Two sample implementations of the parallel algorithm are discussed, one for low-latency cheap communication and another for high-latency expensive communication. The low-latency algorithm is implemented as part of the Data Assimilation Research Testbed facility at NCAR, and scaling results are presented for large models using this algorithm.

The parallel algorithm can also be advantageous for single-processor implementations. It may be expensive in large, complicated models to compute forward observation operators individually as required by the sequential algorithm. For instance, observations of 6-h accumulated precipitation could technically require rerunning an ensemble of model integrations for 6-h before computing each rainfall forward operator. The parallel algorithm allows all forward operators for the same time to be computed at once. The algorithm can even be extended to allow observation operators from different times to be computed at one time. This capability is already a central feature of a number of other filter implementations where it is primarily important for time interpolation of operators (Houtekamer et al. 2005; Houtekamer and Mitchell 2005; Ott et al. 2004; Hamill and Whitaker 2005).

Section 2 reviews the least squares framework for assimilation, while section 3 reviews the sequential algorithm and introduces the parallel algorithm. Section 4 outlines two implementations of the parallel algorithm and discusses computational cost, and section 5 presents conclusions.

2. A probabilistic framework for data assimilation

The ensemble filter assimilation methods described in A03 are Monte Carlo approximations to a general filtering algorithm developed using Bayes theorem (Tarantola 1987). The probability distribution of a model state is approximated by an N-member sample (ensemble) of M-dimensional state vectors,

$$\mathbf{x}_n,\ n = 1, \ldots, N, \tag{2.1}$$

where N is the ensemble size, and each $\mathbf{x}_n$ is an M vector. The set of all observations available by time t_{a-1} is $\mathbf{Y}_{a-1}$ and an additional set of K scalar observations, $\mathbf{y}_a = \{y^1, y^2, \ldots, y^K\}$ becomes available at t_a. Ensemble filters compute an updated (posterior) sample at t_a, $p(\mathbf{x}, t_a|\mathbf{Y}_{a-1}, \mathbf{y}_a)$, given the posterior distribution at t_{a-1}, $p(\mathbf{x}, t_{a-1}|\mathbf{Y}_{a-1})$.

First, posterior samples at t_{a-1} are advanced to time t_a with a model

$$\begin{aligned}\mathbf{x}_{a,n} &= F(\mathbf{x}_{a-1,n}, t_{a-1}, t_a)\\ &= f(\mathbf{x}_{a-1,n}, t_{a-1}, t_a) + g(\mathbf{x}_{a-1,n}, t_{a-1}, t_a),\\ n &= 1, \ldots, N,\end{aligned} \tag{2.2}$$

where f is a deterministic function and g is a stochastic function in which a different realization of the stochastic components is used to advance each ensemble member. The first subscript on $\mathbf{x}$ in (2.2) indexes the time and the second the ensemble member. The time-integrated sample (2.2) is an ensemble approximation of the prior distribution at t_a, $p(\mathbf{x}, t_a|\mathbf{Y}_{a-1})$. (Throughout this article, k, m, and n are reserved to index observations, state variable components, and ensemble members, respectively.)

Observations in $\mathbf{y}_a$ are related to the model state vector by forward observation operators,

$$y^k = h_k(\mathbf{x}) + \upsilon_k, \quad k = 1, \ldots, K, \tag{2.3}$$

where the observational error distribution, a function of both the observing system and the model, through representativeness error (Hamill and Whitaker 2005), is

$$\upsilon_k = \text{Normal}(0, \sigma^2_{o,k}). \tag{2.4}$$

If observational errors in (2.3) are independent of all previous observations, an application of Bayes theorem leads to

$$p(\mathbf{x}, t_a|\mathbf{Y}_{a-1}, \mathbf{y}_a) = p(\mathbf{y}_a|\mathbf{x})p(\mathbf{x}, t_a|\mathbf{Y}_{a-1})/\text{norm}. \tag{2.5}$$

The denominator is a normalization so that (2.5) is a probability distribution but is not explicitly computed in the ensemble algorithms described here.

Further assuming that the υ_k for the observations in set $\mathbf{y}_a$ are mutually independent allows sequential assimilation of observations (Houtekamer and Mitchell 2001)

$$p(\mathbf{x}, t_{a,k+1}) = p(y^k|\mathbf{x})p(\mathbf{x}, t_{a,k})/\text{norm}, \quad k = 1, \ldots, K, \tag{2.6}$$

with

$$p(\mathbf{x}, t_{a,k}) \equiv p(\mathbf{x}, t_a|\mathbf{Y}_{a-1}, \{y^j, j < k\}), \quad k = 1, \ldots, K+1. \tag{2.7}$$

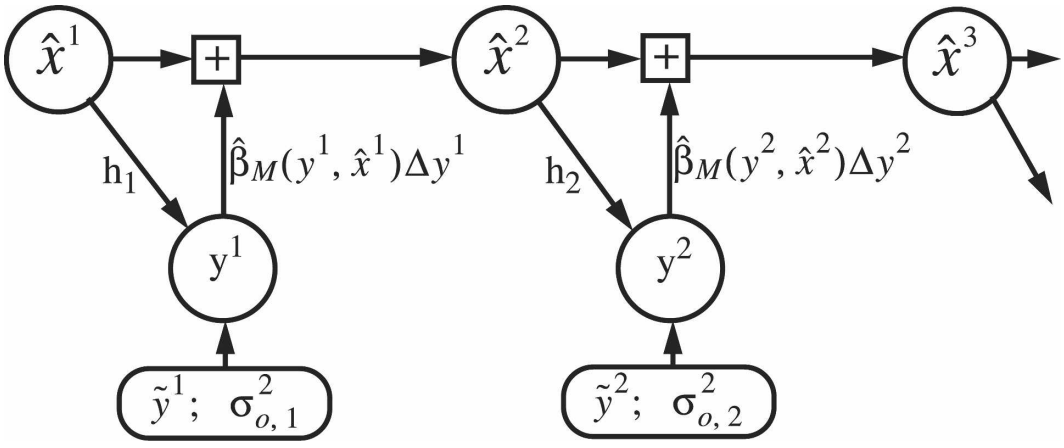

FIG. 1. A schematic depiction of the sequential filter algorithm. The forward observation operator for the first observation, h_1, is applied to the ensemble state vector, $\hat{\mathbf{x}}^1$, to produce a prior ensemble approximation, $\mathbf{y}^1$, of the observation. Observation increments, $\Delta\mathbf{y}^1$, are computed using the observation value, $\tilde{y}^1$, and error variance, $\sigma^2_{o,1}$, and regression is used to compute increments for the state, $\hat{\beta}_M(\mathbf{y}^1, \hat{\mathbf{x}}^1)\Delta\mathbf{y}^1$. The state is updated by adding the increments to produce $\hat{\mathbf{x}}^2$ and the process is repeated for each observation in turn.

Equation (2.6) computes the posterior distribution after assimilating the kth observation in $\mathbf{y}_a$. This posterior is the product of the observation likelihood (first term in the numerator) times the prior distribution conditioned on all observations before the kth (second term in the numerator). Given the assumptions above, an ensemble method for solving (2.6) along with a model to advance state estimates in time (2.2) is sufficient for building an ensemble filter. Ensemble methods can be applied sequentially to assimilate any number of sets of observations.

3. Two ensemble filter algorithms

a. Sequential algorithm

Figure 1 is a schematic depiction of the A03 algorithm showing how observations available at a single time can be assimilated sequentially. The kth observation is assimilated by applying the forward observation operator h_k to each ensemble sample of the state vector $\hat{\mathbf{x}}^k$ that has been updated by all observations before the kth; $\hat{\mathbf{x}}^k$ is an ensemble representation of (2.7) for a given k. The hat on the state vector in Fig. 1 serves to distinguish these from state estimates in the parallel algorithm (Figs. 2 and 3). This yields a prior ensemble estimate of the observation y^k

$$y^k_n = h_k(\hat{\mathbf{x}}^k_n), \quad n = 1, \ldots, N, \tag{3.1}$$

where n indexes the ensemble member, and each $\hat{\mathbf{x}}^k_n$ is an M vector.

The algorithm next computes an ensemble of increments, Δy^k, for y^k given the prior ensemble, the observed value $\tilde{y}^k$, and the observational error variance $\sigma^2_{o,k}$. [The use of $\tilde{y}^k$ instead of y^o_k and other deviations from the recommendations in Ide et al. (1997) are an attempt to clarify the notation used for the parallel ensemble algorithm below.]

For instance, the Ensemble Adjustment Filter (Anderson 2001) computes increments by approximating y^k as Normal($\bar{y}^k$, σ^2_k), where $\bar{y}^k$ and σ^2_k are the ensemble mean and variance. The product of Normal($\bar{y}^k$, σ^2_k) and the observation likelihood, here assumed to be Normal($\tilde{y}^k$, $\sigma^2_{o,k}$), in the numerator of (2.6) is Normal($\bar{y}^u$, σ^2_u) with

$$\sigma^2_u = [(\sigma^2_k)^{-1} + (\sigma^2_{o,k})^{-1}]^{-1} \tag{3.2}$$

and

$$\bar{y}^u = \sigma^2_u[(\sigma^2_k)^{-1}\bar{y}^k + (\sigma^2_{o,k})^{-1}\tilde{y}^k]. \tag{3.3}$$

The observation prior ensemble is shifted and linearly compacted giving an updated ensemble with sample statistics $\bar{y}^u$ and σ^2_u. Increments are

$$\Delta y^k_n = \sqrt{\sigma^2_u/\sigma^2_k}(y^k_n - \bar{y}^k) + \bar{y}^u - y^k_n, \quad n = 1, \ldots, N. \tag{3.4}$$

Many other ensemble filters algorithms differ only in the details of the way in which these increments are computed. Computation of the observation space increments for the kth observation can be written

$$\Delta\mathbf{y}^k = D(\mathbf{y}^k, \tilde{y}^k, \sigma^2_{o,k}), \tag{3.5}$$

where D is an N-vector function and $\mathbf{y}^k$ and $\Delta\,\mathbf{y}^k$ are vectors containing all ensemble members for the prior

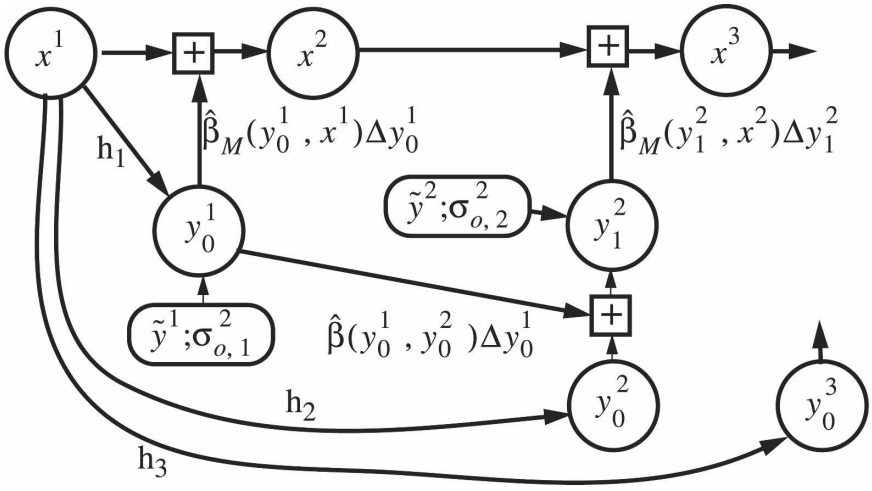

FIG. 2. A schematic depiction of the parallel filter algorithm. All forward observation operators, h_1, h_2, h_3, are computed to produce prior ensemble estimates, $\mathbf{y}_o^1$, $\mathbf{y}_o^2$, $\mathbf{y}_o^3$, of the observations. Increments, $\Delta\mathbf{y}_o^1$, for the first observation prior ensemble are computed. Regression is used to compute increments for the state variables and for each of the subsequent observation priors, y_o^2, y_o^3, etc. The state and the subsequent observation ensembles are updated and the process is repeated for the second observation.

and the increments, respectively. The perturbed observation approach of the ensemble Kalman filter (Houtekamer and Mitchell 1998; Burgers et al. 1998) is an example of an algorithm that uses an alternate increment function, D. Other ensemble filter methods (Tippett et al. 2003; Pham 2001) are also directly compatible with both the parallel and sequential algorithms developed in this section since D can be replaced by an alternative without impacting the rest of the algorithms.

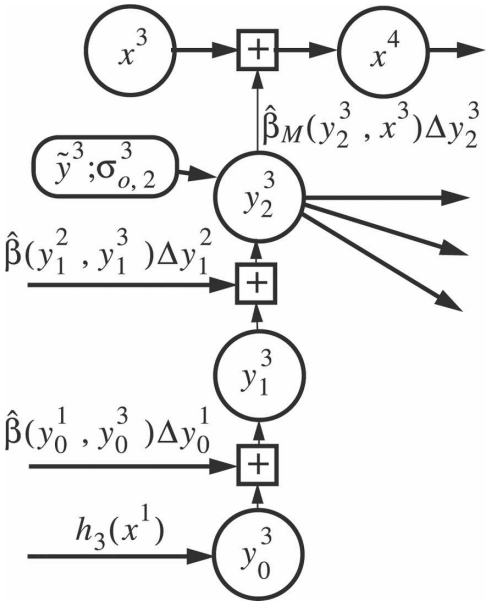

FIG. 3. Details of the parallel filter algorithm's use of the third observation available at a particular time.

Increments for each prior state vector component are computed by linearly regressing using the prior joint sample statistics. The increments from the kth observation are

$$\Delta\hat{x}^k_{m,n} = \hat{\beta}(\mathbf{y}^k, \hat{\mathbf{x}}^k_m)\Delta y^k_n, \quad m = 1, \ldots, M, \quad n = 1, \ldots, N, \tag{3.6}$$

where $\mathbf{y}^k$ and $\hat{\mathbf{x}}^k_m$ are ensemble size vectors. The sample regression operator is

$$\hat{\beta}(\mathbf{a}, \mathbf{b}) \equiv \sum_{n=1}^{N} (a_n - \overline{a})b_n \Big/ \sum_{n=1}^{N} (a_n - \overline{a})^2 = \sum_{n=1}^{N} (a_n - \overline{a})b_n \Big/ S(\mathbf{a}), \tag{3.7}$$

where $\mathbf{a}$ and $\mathbf{b}$ are N vectors and the bar is an ensemble mean.

It is convenient to define the M vector of regression coefficients,

$$\hat{\beta}_M(\mathbf{y}^k, \hat{\mathbf{x}}^k) = [\hat{\beta}(\mathbf{y}^k, \hat{\mathbf{x}}^k_1), \hat{\beta}(\mathbf{y}^k, \hat{\mathbf{x}}^k_2), \ldots, \hat{\beta}(\mathbf{y}^k, \hat{\mathbf{x}}^k_M)], \tag{3.8}$$

where $\hat{\mathbf{x}}^k_m$ is an N vector composed of the ensemble samples of the mth component of the state vector.

Finally, the increments in (3.6) are added to $\hat{\mathbf{x}}^k$ to generate $\hat{\mathbf{x}}^{k+1}$, represented by the plus in Fig. 1,

$$\hat{x}^{k+1}_{m,n} = \hat{x}^{k}_{m,n} + \Delta\hat{x}^{k}_{m,n}, \quad m = 1, \ldots, M, \quad n = 1, \ldots, N. \tag{3.9}$$

The steps above are then repeated for each subsequent observation until all have been used.

b. Parallel algorithm

Figure 2 depicts a parallel algorithm for ensemble filters. First, the prior ensembles for *all* observations at this time are computed (see also Houtekamer et al. 2005; Anderson 2001) by applying the forward observation operators to each ensemble sample of $\mathbf{x}^1$ (three observations are shown in Fig. 2):

$$y^{k}_{0,n} = h_k(\mathbf{x}^1_n), \quad n = 1, \ldots, N. \tag{3.10}$$

Additional notation is required for the parallel algorithm. Terms like $\mathbf{y}^k_j$ (an N vector) are ensemble samples of the kth observation after it has been impacted by the first j observations; $y^k_{0,n}$ in (3.10) refers to the nth ensemble member of the kth observation before it has been impacted by any other observations.

The algorithm computes increments, $\Delta\mathbf{y}^1_o$, for the first observation and uses regression to update both the state variable estimate, $\mathbf{x}^1$, and the prior ensemble estimates for *all* subsequent observations, $\mathbf{y}^2_o$, $\mathbf{y}^3_o$, . . . , as shown in Figs. 2 and 3. Next, the updated prior, $\mathbf{x^2}$ in Fig. 2, observed value, $\tilde{y}^2$, and observational error variance, $\sigma^2_{o,2}$, for the second observation are used to compute increments. Regression is used to update the state estimate, $\mathbf{x^2}$, and the prior ensembles for the third and all subsequent observations. The process is repeated for the prior ensemble of the third observation updated by the first and second observations as shown in Fig. 3. Again, the prior, observed value and observation variance are used to compute increments for the third observation and regressed increments are computed for the state and for the priors for observations 4 and above. The procedure is repeated until increments for all observations have been computed and applied via regression.

A more formal inductive description of the parallel algorithm follows. Assume that increments for all observations before the kth have been computed and used so that y^k_{k-1}, the prior ensemble estimate of the kth observation conditioned on all previous observations, is available. Increments for $\mathbf{y}^k_{k-1}$ are computed as

$$\Delta\mathbf{y}^k_{k-1} = D(\mathbf{y}^k_{k-1}, \tilde{y}^k, \sigma^2_{o,k}), \tag{3.11}$$

where D is defined in (3.5). Increments for the state estimates are computed as

$$\Delta x^k_{m,n} = \hat{\beta}(\mathbf{y}^k_{k-1}, \mathbf{x}^k_m)\Delta y^k_{k-1,n}, \quad m = 1, \ldots, M, \quad n = 1, \ldots, N \tag{3.12}$$

and added to the state vector ensemble $\mathbf{x}^k$.

Increments due to the assimilation of the kth observation are also computed and added to all subsequent observation prior ensemble estimates:

$$y^j_{k,n} = y^j_{k-1,n} + \hat{\beta}(\mathbf{y}^k_{k-1}, \mathbf{y}^j_{k-1})\Delta y^k_{k-1,n}, \quad j = k+1, \ldots, K, \quad n = 1, \ldots, N. \tag{3.13}$$

The algorithm repeats until all K observations have been used.

c. Comparison of sequential and parallel algorithms

The key modification in the parallel algorithm is that the observation prior ensembles are updated exactly like the state variables by regression using sample statistics. This entails added computation to do the regressions of each observation variable's increments onto all subsequent observations at a cost of $O(K^2)$.

Even on a single processor, the parallel algorithm can lead to substantial computational savings and simplified implementation when the formulation of models makes it easier to apply forward observation operators all at once, rather than sequentially after intermediate state updates. This is also used to do time interpolation of forward observation operators in many ensemble filtering systems (Houtekamer et al. 2005; Ott et al. 2004). A good example involves the assimilation of 6-h accumulated rainfall observations in an atmospheric GCM. Rainfall is generally the sum of an output of a physical parameterization package over a sequence of model time steps. In the sequential algorithm, this requires that the ensemble of model advances be rerun over the 6-h interval in order to compute a prior ensemble of rainfall values for each individual rainfall observation. This is clearly impractical in large models. More subtle examples occur when conversion of model state variables to another grid or a nonstate variable type are part of forward operators. In such cases, computing a large set of observations at once may be much cheaper than computing them independently.

d. Localization of observation impact

A number of enhancements are used to improve ensemble filter performance. Errors in ensemble filters tend to produce a systematic underestimate of ensemble variance that can lead to poor filter perfor-

mance or filter divergence (Chui and Chen 1987). This can be ameliorated by covariance inflation (Anderson and Anderson 1999), which adds variance to prior ensemble estimates by linearly "inflating" ensemble members around the ensemble mean. Application of inflation is identical in the sequential and parallel algorithms. Inflation is applied immediately after the ensemble members are advanced by the model but before assimilation begins.

The ensemble sample regression coefficients in (3.12 and 3.13) are subject to sampling error that increases as the absolute value of the correlation between an observation and a state variable (or another observation) becomes small. The amount of information available from an observation that is weakly correlated with a state variable is also expected to be small. To reduce error and computation, the regression coefficients in (3.12 and 3.13) can be multiplied by a function of the expected correlation between the observation and the state (or other observed) variable. If the expected absolute value of the correlation is small enough, this weight can be set to zero and the regression need not be performed. The expected correlation is not generally known a priori. However, in many dynamical models there is reason to believe that the correlations become smaller as a function of the physical distance from the observation (Mitchell and Houtekamer 2000; Hamill et al. 2001). When the regression of increments from an observation located at z_o onto a state variable (or observation prior) at z_s is performed with (3.12, 3.13), the regression coefficient can be multiplied by $\zeta(d, c)$, where d is the distance between z_s and z_o, and c specifies how rapidly the correlation is believed to fall off as a function of distance. In the atmospheric and ocean literature, the most common ζ is the piecewise continuous compactly supported function of Gaspari and Cohn (1999) with half-width c. For $d \geq 2c$, the observation has no impact on the state variable. For $d < 2c$, ζ approximates a Gaussian. More recently, algorithms for adaptively computing the required localization (Anderson 2007) have been developed and these are also compatible with both the parallel and sequential algorithm.

4. Implementations of the parallel algorithm

This section examines implementing the parallel algorithm on P processors. Here K is the number of observations available at a given time, M is the size of the state vector, and N is the ensemble size.

Although implementing the parallel algorithm on a single processor requires additional computation, it is more scalable than the sequential one. When ensemble members are being advanced in time, the entire state vector for one ensemble is on a single (set of) processors; this is referred to as state-complete. When computing observation increments and regressions during the assimilation step, all ensemble members of a given state vector element are on a single processor so that ensemble means and variances can be computed without communication; this is referred to as ensemble-complete. Transposing ensembles of state vectors between state-complete and ensemble-complete requires communicating $O(MN)$ real values. The sequential algorithm is not amenable to a multiprocessor implementation in this way; direct implementation would require a pair of transposes for the assimilation of each scalar observation.

The parallel algorithm allows all forward operators for a set to be computed in parallel immediately after the model advance while storage is state-complete (Fig. 2). A single transpose to ensemble-complete is required before assimilating. In this case, the observation prior ensembles must also be transposed to ensemble-complete, necessitating communication of an additional $O(KN)$ reals.

Both implementations also require advancing models and computing forward observation operators. Model advances are easily parallelized up to the number of ensemble members, N, and to QN if a single model integration can be parallelized across Q processors (although this implicitly involves additional communication). Forward observation operators are trivially parallel across N processors but may require additional communication to be parallelized on more than N processors.

a. Low-latency implementation

Details of interprocessor communication capabilities can motivate a variety of distributions of ensemble-complete state vector components and observation priors on processors. The first implementation of the parallel algorithm assumes that communication latency is low and speed is high and proceeds as follows:

1) Model ensembles are advanced to the time of the next observation set.
2) Each of the K forward observation operators is applied to each of the N ensemble estimates (Fig. 2).
3) The state ensembles and the observation prior ensembles are transposed to ensemble-complete. Each processor gets M/P randomly selected state vector components and K/P randomly selected observation priors. The observed value and observational error

standard deviation are placed on the processor with the corresponding prior observation ensemble.

4) For each observation in turn:
   (i) The processor with this observation prior computes the observation increments (3.11) at cost $O(N)$ and broadcasts the observation prior distribution and the increments, $O(N)$ reals.
   (ii) Each processor uses the increments to update each of its state vector (3.12) and observation prior (3.13) ensembles. Number of operations per processor is $O(NM/P + NK/P)$. This step scales to a total of $M + K$ processors if each has only a single ensemble of a state variable or observation prior.
5) The updated state vector ensembles are transposed to state-complete by communicating of $O(NM)$ reals.

The rate-limiting step in the assimilation portion is the sequential computation of the observation increments, a total of $O(KN)$ operations, plus the communication of $O(KN)$ reals. If localization is being used, it is possible to compute prior increments in parallel for observations that are separated by distance $d > 2c$. As observation density gets large, one could also adaptively reduce the cutoff distance c as a method of observation "thinning" (Bormann et al. 2003). This would allow more observation increments to be computed in parallel as the number of observations increased.

To minimize waiting, when the information for observation k is broadcast, the processor with the prior distribution for observation $k + 1$ can first compute the update for observation $k + 1$ and broadcast the increments before updating the rest of its state variable components and observations. The random distribution of the state vector components and observations on processors addresses issues related to load balancing if the number of state variables and observations close to a given observation is not uniform and if localization is being used to limit the impact of an observation to some "nearby" region.

b. A very high-latency implementation

If the latency cost associated with communications is very large, a different implementation of the parallel algorithm can allow more limited speedup on smaller numbers of processors. This algorithm can be useful on commodity clusters in which communication between processors is through a slow interconnect or a shared file system. It assumes that any communication between processors outside of the transposes is prohibitively expensive.

1) and 2) Same as the low-latency implementation.
3) In the transpose, each processor is given ensembles for a set of state variable components that are "close" to each other for localization purposes. For simplicity, assume that load balancing is not an issue and each processor receives M/P state variable components. Each processor also receives the observation prior ensembles for all observations that are close to any of its state variables; the number is bounded above by K.
4) Each processor sequentially processes its set of observations as follows:
   (i) The observation increments are computed at cost of $O(N)$.
   (ii) Each state variable component and unassimilated observation prior on the processor are updated by regression at cost $O(NM/P + NK/P)$.

   The total cost for step 4 is $O(KN + KNM/P + NK^2/P)$.
   5) Reverse transpose is identical to low-latency implementation.

Load-balancing problems can exist if the observation density is not uniform across the state variable domains assigned to different processors in step 3. More sophisticated implementations could have fewer model state variables on a processor if they were associated with larger numbers of observations. For instance, in an NWP application with in situ observations, larger sets of variables from over the southern ocean would be placed on a single processor as compared to state variables from over the much more densely observed North America.

c. Bit-wise reproducibility

The low-latency implementation of the parallel algorithm can be coded so that the results are reproducible no matter how the problem is partitioned on an arbitrary number of processors and for arbitrary ensemble initial conditions and observation operators. The observations are always processed in the same order, so the observation prior increments are always identical. For stochastic update algorithms like the perturbed observations ensemble Kalman filter (Houtekamer and Mitchell 1998; Burgers et al. 1998) care must be taken so that the same random sequence is used no matter the number of processors; in the Canadian system this is done using seeds that are a function of the observation location, ensemble number, and the date. The impact of the increments on the state variables and the subsequent observation prior ensembles can be computed completely in parallel so the partitioning onto proces-

sors has no impact on the results. This bit-wise reproducibility can be helpful for validating the parallel algorithm. When applied to models that depend sensitively on initial conditions, even truncation errors can lead to large differences as an assimilation proceeds. It can be difficult to distinguish these differences from errors in the implementation; this is not an issue with the low-latency implementation.

The high-latency algorithm is only guaranteed to be bit-wise reproducing if no localization is used. Without localization, each processor computes the observation increments for every observed variable and these can be done in the same order on each processor. If localization is used, a given processor need only compute increments for observations that are "close" to one of the state variables on that processor. In essence, each processor assimilates the observations for a given time in a different order, doing first those that are close to its state variables. The observations that are not close can be viewed as being assimilated after the close ones, but these computations need not be completed since they will not impact any state variables. For all algorithms outlined in section 2 that assume a linear relation between prior distributions of observations and state variables, the order in which observations from a given time are assimilated is irrelevant. However, this is not the case when implemented with localization and in the presence of truncation error from finite precision arithmetic. Neighboring state variables that are on different processors for the assimilation step could end up with inconsistencies. However, results using up to 16 domains with an atmospheric general circulation model [the National Center for Atmospheric Research's (NCAR) Community Atmosphere Model version 3.1 (CAM 3.1) at T85 horizontal resolution and 26 levels (Collins et al. 2004)] show that the domain boundaries do not display evident discontinuities when assimilating the observations used in the NCAR–National Centers for Environmental Prediction (NCEP) reanalysis.

d. Hybrid implementations

Various hybrids of the algorithms and implementations can be devised to reduce computation time on a given computing platform. For instance, a hybrid of the sequential and parallel algorithms is possible. The set of state variable ensembles on a given processor during the assimilation may allow on-processor computation of forward observation operators for a subset of the observations. These observations could be processed as in the sequential algorithm. For an observation of this type, the forward operator for each ensemble member could be computed when it was time to assimilate the observation. There would be no need to compute the forward observation operator initially as in the parallel algorithm, and no need to regress the increments from previous observations onto an ensemble distribution for this observation. It is efficient to process these observations last on this processor since there is then no need to regress the observation increments onto any of the observations whose forward operators cannot be computed on processor (they will already have been assimilated). If the number of observations for which forward observation operators can be computed on a given processor is large compared to the total number of observations on the processor, this can be a substantial savings.

Another hybrid splits the set of observations available at a given time into a number of subsets; for simplicity, assume that the K observations are split into S equal subsets. Each subset is treated as if it were all the observations available at a given time. After the assimilation of a subset, a transpose is used to get state-complete data. The forward observation operators for the next subset are computed, and a transpose to ensemble-complete precedes the assimilation of this subset. The number of transpose pairs is increased from 1 to S. However, the expected cost of updating the observation prior ensembles on a given processor over all the subsets is reduced by a factor S from $O(NK^2/P)$ to $O(NK^2/PS)$. In the limit of K subsets, this becomes the sequential algorithm with transposes. For a given observation set size, number of processors, and transpose cost, an optimal number of subsets can be computed to minimize wall-clock computation time.

5. Results

Most users of NCAR's Data Assimilation Research Testbed (DART) use Linux clusters with relatively high-speed interconnects for their large computing jobs at present. These users are interested in having assimilation algorithms that scale well over a range of processors. Another key aspect of DART is to provide generic assimilation tools for many different types of geophysical models. The low-latency algorithm requires no knowledge about the layout of the model state vector. This means that model developers do not need to find ways to communicate information about the metadata of their state vector to the assimilation.

The low-latency algorithm has been run on several different Linux clusters with up to 64 processors. The approximately 250 000 observations every 12 h used in the NCEP–NCAR reanalysis for January 2003 have been assimilated in four different models: a spectral

version of NCAR's CAM, a finite-volume version of CAM, the Weather Research and Forecasting (WRF) regional forecasting model with a North American domain, and the B-grid dynamical core of the Geophysical Fluid Dynamics Laboratory Atmosphere Model version 2 (GFDL AM-2).

Performance was evaluated on two Linux clusters. The first was NCAR's Lightning Linux cluster, an IBM SMP running SuSE Enterprise Server 9. Each node has two 2.2-GHz Opteron processors and 4-GB shared memory. The interconnect fabric is a 128-port Myrinet switch. The code was compiled with PathScale FORTRAN version 2.4 and the MPI library was MPICH version 1.2.6. The second was NCAR's Coral cluster, an Aspen Systems Linux SMP running Suse 9.2. Each node has two 3.2 GHz IA-32 EM64T nodes and 4-GB shared memory. The interconnect fabric is an Infiniband switch and code was compiled with Intel FORTRAN 9.0 and MPICH version 1.2.5. Timing results below only include the assimilation and do not include the model advances since these should be embarrassingly parallel.

As an example, Fig. 4 shows normalized wallclock times,

$$nT_n/bT_b, \tag{5.1}$$

for 32 ensemble members assimilations with T21, T42, and T85 versions of the spectral CAM with 26 levels; the state vector lengths are approximately 300 000, 1.3 million, and 5.1 million. In (5.1), T_n is the total wall-clock time for the computation with n processors and T_b is the time taken (by the first experiment performed) on b processors. For T21, b is 1 and for T42 and T85 b is 4, the smallest number of processors on which the job would run due to memory constraints. Four cases were run for each T21 processor count and two for each T42 and T85 count to give a minimal estimate of run to run variability in scaling. The number of state variables per processor varies from approximately 300 000 to 9000 for T21, 325 000 to 20 000 for T42, and 1.3 million to 80 000 for T85. Throughout this range, the normalized wall clock stays very close to 1. For all three resolutions, the normalized time rises as processor count is increased initially, consistent with the increased cost to broadcast the observation priors in step 4(i) of the algorithm and the increased communication cost of doing state transposes with larger processor counts. However, all three resolutions show a dip in the normalized time for larger processor counts, at 4 processors for T21, 32 for T42, and 64 for T85. These better than linear speedups appear to be due to increased cache coherency as the size of the state vector ensemble stored on each processor during the assimilation decreases. For the T21 and T42, normalized time begins to increase again for larger processor counts as communication efficiency is reduced. Similar scaling behavior was found for all four models on both clusters with a variety of compiler options.

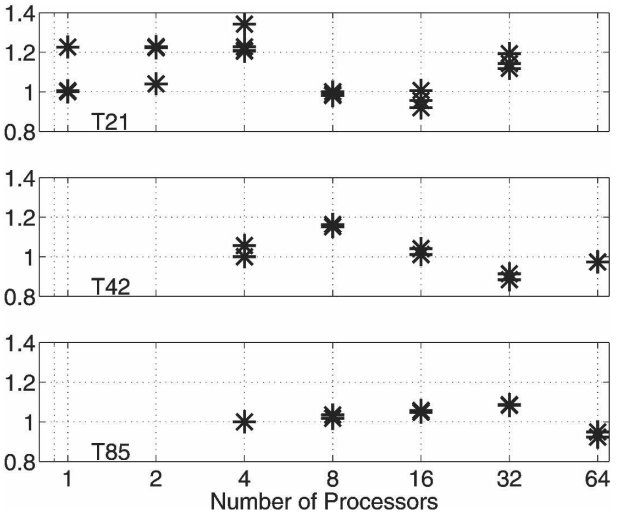

FIG. 4. Normalized wall-clock time for the assimilation portion of assimilations of observations used in the NCEP–NCAR reanalysis for January 2003 using the low-latency parallel algorithm. Assimilations are done with the spectral CAM model at (top) T21, (middle) T42, and (bottom) T85 with 26 vertical levels. Normalized wall-clock time is relative to the single processor results for T21 and the four processor results for T42 and T85. A value less than 1 indicates greater than linear speedup, while values greater than 1 are less than linear speedup. Four cases are plotted for each T21 processor count and two for each of the T42 and T85 results.

6. Summary

Two scalable implementations of a parallel algorithm for doing ensemble filter assimilation have been described in the least squares framework of A03. For linear forward observation operators, the parallel algorithm is equivalent to the sequential algorithm in A03. The low-latency implementation of the parallel algorithm can be coded to produce bit-wise identical results for any distribution of ensemble priors on any number of processors. The high-latency implementation produces bit-wise identical answers on an arbitrary number of processors; when localization is applied, results are found to be qualitatively similar but may differ quantitatively as the number of processors is varied.

The high-latency implementation of the parallel algorithm requires careful attention to the way in which state variables are distributed on processors during the assimilation phase of the algorithm. However, the low-

latency parallel algorithm does not require careful design of compute domains or computational halos as in other proposed parallel filter implementations. It scales to a large number of processors and is tolerant to load imbalances due to spatially varying observational density. In this sense it is a relatively generic algorithm that can be implemented efficiently without detailed knowledge about the model's state variables or the forward operators.

The low-latency parallel algorithm implementation has been evaluated in a number of low-order model assimilations and for a large climate model (NCAR's CAM 3.1 at T85 resolution with 26 vertical levels) assimilating the observations used in the NCEP–NCAR reanalysis (Kistler et al. 2001). Results for an arbitrary number of domains are identical and speedup is consistent with the analysis in section 4. Implementations of the high-latency algorithm with localization do not produce identical results but function qualitatively correctly. The various implementations of the parallel filter algorithm should provide recipes for good performance across a wide array of parallel machines. For machines with a large number of commodity processors with a high-speed custom interconnection network, the current architecture of choice for operational numerical weather prediction, the low-latency implementation should be effective. For large models with large observation sets (as in Houtekamer et al. 2005), good scaling should be attainable for many thousands of processors. Versions of the parallel algorithm implementations have been incorporated in NCAR's Data Assimilation Research Testbed and are available for community evaluation (http://www.image.ucar.edu/DAReS/DART; the high-latency algorithms are available in the Iceland release while the low-latency algorithms are in the Jamaica and later releases).

Acknowledgments. The authors thank Tim Hoar, Kevin Raeder, and Hui Liu for their work on the DART system. Thanks to three anonymous reviewers for comments that have significantly improved this work.

APPENDIX

Conditions for which Sequential/Parallel Produce Identical Results

Let all forward observation operators after the first be linear,

$$h_k(x) = \mathrm{A}_k^{\mathrm{T}} x = \sum_{j=1}^{M} a_{k,j}\mathbf{x}_j, \quad k = 2, \ldots, K. \quad \text{(A.1)}$$

An inductive proof demonstrates that the sequential and parallel algorithms are identical for *arbitrary* initial state ensembles $\hat{\mathbf{x}}^1 = \mathbf{x}^1$. Suppose that the first k ensemble state estimates are identical [Eqs. (A.2), (A.3), and (A.4) compare the parallel algorithm on the lhs with the sequential on the rhs],

$$\mathbf{x}^j = \hat{\mathbf{x}}^j, \quad j \le k, \quad \text{(A.2)}$$

and note that this is true for $k = 2$ since the impacts of the first observation on the initial state estimate are identical in both algorithms. Also suppose that the final prior estimates of the observed variables are identical for the first $k - 1$ observations,

$$y^j_{j-1,n} = y^j_n, \quad j \le k - 1, \quad n = 1, \ldots, N, \quad \text{(A.3)}$$

which also implies that the observation prior increments are the same:

$$\Delta y^j_{j-1,n} = \Delta y^j_n, \quad j \le k - 1, \quad n = 1, \ldots, N. \quad \text{(A.4)}$$

Suppose also that

$$y^k_{j-1,n} = \sum_{m=1}^{M} a_{k,m}\hat{x}^j_{m,n}, \quad n = 1, \ldots, N, \quad \text{(A.5)}$$

which is true for $j = 1$ since (A.2) implies that

$$A_k^{\mathrm{T}}\hat{\mathbf{x}}^1 = \mathbf{y}^k_0. \quad \text{(A.6)}$$

The lhs of (A.5) is the ensemble prior estimate for the kth observation conditioned on all previous observations in the parallel algorithm while the rhs is the result of applying the kth forward observation operator to $\hat{\mathbf{x}}^j$ (which is not actually performed in the sequential algorithm unless $k = j$).

Following Fig. 3 for the parallel algorithm and using (A.4),

$$\begin{aligned} y^k_{j,n} &= y^k_{j-1,n} + \hat{\beta}(\mathbf{y}^j_{j-1}, \mathbf{y}^k_{j-1})\Delta y^j_{j-1,n}, \\ &= y^k_{j-1,n} + \hat{\beta}(\mathbf{y}^j, \mathbf{y}^k_{j-1})\Delta y^j_n, \\ n &= 1, \ldots, N. \end{aligned} \quad \text{(A.7)}$$

The increment is

$$y^k_{j,n} - y^k_{j-1,n} = \frac{\Delta y^j_n}{S(\mathbf{y}^j)} \sum_{i=1}^{N} [(y^j_i - \bar{y}^j)y^k_{j-1,i}], \quad n = 1, \ldots, N. \quad \text{(A.8)}$$

Following Fig. 1 for the sequential algorithm,

$$\hat{x}^{j+1}_{m,n} = \hat{x}^j_{m,n} + \hat{\beta}(\mathbf{y}^j, \hat{\mathbf{x}}^j_m)\Delta y^j_n, \quad n = 1, \ldots, N \quad \text{(A.9)}$$

and

$$\sum_{m=1}^{M} a_{k,m}\hat{x}^{j+1}_{m,n} = \sum_{m=1}^{M} a_{k,m}\hat{x}^{j}_{m,n} + \sum_{m=1}^{M} [a_{k,m}\hat{\beta}(\mathbf{y}^j, \hat{\mathbf{x}}^j_m)\Delta y^j_n], \quad n = 1, \ldots, N. \tag{A.10}$$

The increment is

$$\sum_{m=1}^{M} a_{k,m}\hat{x}^{j+1}_{m,n} - \sum_{m=1}^{M} a_{k,m}\hat{x}^{j}_{m,n} = \frac{\Delta y^j_n}{S(\mathbf{y}^j)} \sum_{m=1}^{M} \left\{ a_{k,m} \sum_{i=1}^{N} [(y^j_i - \bar{y}^j)\hat{x}^j_{m,i}] \right\}, \quad n = 1, \ldots, N. \tag{A.11}$$

Reordering the sums gives

$$\sum_{m=1}^{M} a_{k,m}\hat{x}^{j+1}_{m,n} - \sum_{m=1}^{M} a_{k,m}\hat{x}^{j}_{m,n} = \frac{\Delta y^j_n}{S(\mathbf{y}^j)} \sum_{i=1}^{N} [(y^j_i - \bar{y}^j) \sum_{m=1}^{M} (a_{k,m}\hat{x}^j_{m,i})], \quad n = 1, \ldots, N. \tag{A.12}$$

Comparing (A.12) and (A.8) shows that if (A.5) is true for $j = i - 1$, then it is also true for $j = i$. In particular, given (A.6), this shows that $\mathbf{y}^k = \mathbf{y}^k_{k-1}$ and $\Delta\mathbf{y}^k = \Delta\mathbf{y}^k_{k-1}$, which means that $\mathbf{x}^k = \hat{\mathbf{x}}_k$. The two algorithms produce identical results for linear forward observation operators for an arbitrary number of observations being assimilated.

For nonlinear forward observation operators, the algorithms no longer produce identical results. Essentially, they are choosing different places to linearly approximate the nonlinear relation between the state and observation priors. However, no systematic differences have been found between the methods. A more thorough analysis of the differences would be difficult and application specific. When implemented on a computer, the two algorithms may differ in the least significant bits even for linear observation operators. These differences can be amplified by the assimilation of subsequent observations and by the model so actual results may differ even for linear forward observation operators.

REFERENCES

Anderson, J. L., 2001: An ensemble adjustment Kalman filter for data assimilation. *Mon. Wea. Rev.,* **129,** 2894–2903.

——, 2003: A local least squares framework for ensemble filtering. *Mon. Wea. Rev.,* **131,** 634–642.

——, 2007: Exploring the need for localization in ensemble data assimilation using an hierarchical ensemble filter. *Physica D,* **230,** 99–111.

——, and S. L. Anderson, 1999: A Monte Carlo implementation of the nonlinear filtering problem to produce ensemble assimilations and forecasts. *Mon. Wea. Rev.,* **127,** 2741–2758.

Bormann, N., S. Saarinen, G. Kelly, and J.-N. Thépaut, 2003: The spatial structure of observation errors in atmospheric motion vectors from geostationary satellite data. *Mon. Wea. Rev.,* **131,** 706–718.

Burgers, G., P. J. van Leeuwen, and G. Evensen, 1998: Analysis scheme in the ensemble Kalman filter. *Mon. Wea. Rev.,* **126,** 1719–1724.

Chui, C. K., and G. Chen, 1987: *Kalman Filtering: With Real-Time Applications.* Springer, 191 pp.

Collins, W. D., and Coauthors, 2004: Description of the NCAR Community Atmosphere Model (CAM 3.0). NCAR Tech. Note NCAR/TN-464+STR, 226 pp.

Evensen, G., 1994: Sequential data assimilation with a nonlinear quasigeostrophic model using Monte Carlo methods to do forecast error statistics. *J. Geophys. Res.,* **99** (C5), 10 143–10 162.

Gaspari, G., and S. E. Cohn, 1999: Construction of correlation functions in two and three dimensions. *Quart. J. Roy. Meteor. Soc.,* **125,** 723–757.

Hamill, T. M., and J. S. Whitaker, 2005: Accounting for the error due to unresolved scales in ensemble data assimilation: A comparison of different approaches. *Mon. Wea. Rev.,* **133,** 3132–3147.

——, ——, and C. Snyder, 2001: Distance-dependent filtering of background-error covariance estimates in an ensemble Kalman filter. *Mon. Wea. Rev.,* **129,** 2776–2790.

Houtekamer, P. L., and H. L. Mitchell, 1998: Data assimilation using an ensemble Kalman filter technique. *Mon. Wea. Rev.,* **126,** 796–811.

——, and ——, 2001: A sequential ensemble Kalman filter for atmospheric data assimilation. *Mon. Wea. Rev.,* **129,** 123–137.

——, and ——, 2005: Ensemble Kalman filtering. *Quart. J. Roy. Meteor. Soc.,* **131,** 3269–3289.

——, ——, G. Pellerin, M. Buehner, M. Charron, L. Spacek, and B. Hansen, 2005: Atmospheric data assimilation with the ensemble Kalman filter: Results with real observations. *Mon. Wea. Rev.,* **133,** 604–620.

Ide, K., P. Courtier, M. Ghil, and A. C. Lorenc, 1997: Unified

notation for data assimilation: Operational sequential and variational. *J. Meteor. Soc. Japan,* **75B,** 181–189.

Jazwinski, A. H., 1970: *Stochastic Processes and Filtering Theory.* Academic Press, 376 pp.

Kalman, R. E., 1960: A new approach to linear filtering and prediction problems. *Trans. AMSE J. Basic Eng.,* **82D,** 35–45.

Keppenne, C. L., and M. M. Rienecker, 2002: Initial testing of a massively parallel ensemble Kalman filter with the Poseidon isopycnal ocean general circulation model. *Mon. Wea. Rev.,* **130,** 2951–2965.

Kistler, R., W. Collins, S. Saha, G. White, and J. Woolen, 2001: The NCEP–NCAR 50-Year Reanalysis: Monthly means CD-ROM and documentation. *Bull. Amer. Meteor. Soc.,* **82,** 247–267.

Mitchell, H. L., and P. L. Houtekamer, 2000: An adaptive ensemble Kalman filter. *Mon. Wea. Rev.,* **128,** 416–433.

——, ——, and G. Pellerin, 2002: Ensemble size, balance, and model-error representation in an ensemble Kalman filter. *Mon. Wea. Rev.,* **130,** 2791–2808.

Ott, E., and Coauthors, 2004: A local ensemble Kalman filter for atmospheric data assimilation. *Tellus,* **A56,** 415–428.

Pham, D. T., 2001: Stochastic methods for sequential data assimilation in strongly nonlinear systems. *Mon. Wea. Rev.,* **129,** 1194–1207.

Tarantola, A., 1987: *Inverse Problem Theory*. Elsevier, 613 pp.

Tippett, M. K., J. L. Anderson, C. H. Bishop, T. M. Hamill, and J. S. Whitaker, 2003: Ensemble square root filters. *Mon. Wea. Rev.,* **131,** 1485–1490.

Whitaker, J. S., and T. M. Hamill, 2002: Ensemble data assimilation without perturbed observations. *Mon. Wea. Rev.,* **130,** 1913–1924.

Zhang, S., M. J. Harrison, A. T. Wittenberg, A. Rosati, J. L. Anderson, and V. Balaji, 2005: Initialization of an ENSO forecast system using a parallelized ensemble filter. *Mon. Wea. Rev.,* **133,** 3176–3201.