Parallel Implementation of an Ensemble Kalman Filter

P. L. HOUTEKAMER, BIN HE, AND HERSCHEL L. MITCHELL

Data Assimilation and Satellite Meteorology Research Section, Environment Canada, Dorval, Québec, Canada

(Manuscript received 9 January 2013, in final form 12 September 2013)

ABSTRACT

Since mid-February 2013, the ensemble Kalman filter (EnKF) in operation at the Canadian Meteorological Centre (CMC) has been using a 600×300 global horizontal grid and 74 vertical levels. This yields 5.4×10^7 model coordinates. The EnKF has 192 members and uses seven time levels, spaced 1 h apart, for the time interpolation in the 6-h assimilation window. It follows that over 7×10^{10} values are required to specify an ensemble of trial field trajectories. This paper focuses on numerical and computational aspects of the EnKF. In response to the increasing computational challenge posed by the ever more ambitious configurations, an ever larger fraction of the EnKF software system has gradually been parallelized over the past decade. In a strong scaling experiment, the way in which the execution time decreases as larger numbers of processes are used is investigated. In fact, using a substantial fraction of one of the CMC's computers, very short execution times are achieved. As it would thus appear that the CMC's computers can handle more demanding configurations, weak scaling experiments are also performed. Here, both the size of the problem and the number of processes are simultaneously increased. The parallel algorithm responds well to an increase in either the number of ensemble members or the number of model coordinates. A substantial increase (by an order of magnitude) in the number of assimilated observations would, however, be more problematic. Thus, to the extent that this depends on computational aspects, it appears that the meteorological quality of the Canadian operational EnKF can be further improved.

1. Introduction

As available computational facilities become more powerful, it is generally possible to improve the quality of operational data assimilation systems. In the context of an ensemble Kalman filter (EnKF), one may increase the resolution of the model, the ensemble size, and the number of assimilated observations. The increased resolution of the model may itself consist of higher horizontal, vertical, and/or temporal resolution. In recent years, the quality of the operational Canadian global EnKF (Houtekamer et al. 2009, 2014) has improved due, in large part, to changes to all of the above parameters.

Over the last decade, we have witnessed a general change in computer architecture toward systems with more cores. For instance, in 2000, our center had a cluster of NEC SX-4 nodes each with 32 vector processors. At that time, most operational programs could run on just a few processors. Currently, in 2012, our center can make use of a pair of IBM POWER7 clusters. They rank 120 and 121 on the November 2012 top 500 list of the world's top supercomputers (Meuer et al. 2012). Each of these clusters has approximately 8000 cores. Operational programs, running one software thread on each core,[1] can benefit from $O(1000)$ threads. This amounts to an increase of about a factor of 100 over the past 12 yr. The trend toward increasing parallelization, with more cores being available on the fastest computers (Meuer et al. 2012), is likely to continue in the future and represents a significant challenge to data assimilation system developers (Isaksen 2012).

The Canadian EnKF uses the message passing interface (MPI; Gropp et al. 1994) for the communications between processes. Within a single MPI process, several software threads can share the same memory. These threads are activated using OpenMP (Chandra et al.

Corresponding author address: P. L. Houtekamer, Section de la Recherche en Assimilation des Données et en Météorologie Satellitaire, 2121 Route Trans-Canadienne, Dorval, QC H9P 1J3, Canada.
E-mail: peter.houtekamer@ec.gc.ca

DOI: 10.1175/MWR-D-13-00011.1

[1] In this study we did not use the Simultaneous MultiThreading (SMT) option, which permits using two software threads per core on our center's IBM POWER7 computers.

2001). When processes are submitted to have only one software thread, OpenMP directives are ignored. In appendix A, we describe and illustrate how parallel software can be organized to run a number of processes, each consisting of one or more OpenMP threads, on a computer consisting of a number of nodes each having a number of cores.

To obtain state-of-the-art results, the EnKF algorithm necessarily needs to efficiently use the available computer resources. Thus, the EnKF has to make use of disk space, processors, the main computer memory, the memory caches, the job-queuing system, and the archiving system in a manner that permits obtaining results in both a research and development and an operational environment. At Environment Canada, with an operational EnKF featuring 192 members, each using a 600×300 global horizontal grid and 74 vertical levels, we are in a relatively good position to report on computational challenges.

It is not possible for us to implement a number of different candidate EnKF strategies and perform a fair comparative study of their computational and meteorological performance. In any event, there is a strong interdependence between an evolving algorithm and an evolving computational environment. One will, for example, try to adjust the computational procedures when problematic behavior is detected. These adjustments may involve both algorithmic changes and changes to the computational environment. Timing results obtained on the IBM POWER7 clusters of Environment Canada cannot be directly valid in any different computational environment. In this paper, the focus is entirely on the sequential EnKF algorithm whose development was first described in Houtekamer and Mitchell (1998). However, we will note where different design decisions could have been made. We will also mention any algorithmic changes that were implemented to address specific computational bottlenecks. Some of our results may be relevant for other ensemble data assimilation algorithms intended for operational implementation on parallel computers (e.g., Keppenne and Rienecker 2002; Anderson and Collins 2007; Szunyogh et al. 2008; Whitaker et al. 2008).

In section 2, we start with an overview of aspects of the EnKF algorithm that are relevant to understanding its computational performance. In section 3, we discuss requirements of space and memory, which are complementary to the pure speed of execution. In section 4, we focus on the assimilation step of the EnKF and give actual execution times. In section 5, we give scaling results obtained with more processes as well as for bigger problems. Finally, in the discussion, we mention some issues to be addressed in the future.

2. Operational environment and key aspects of the EnKF algorithm

This section contains a brief discussion of the context within which the EnKF operates followed by a discussion of some key aspects of the EnKF algorithm. Many of the latter, including the rationale for perturbing observations and for using subensembles, date back to Houtekamer and Mitchell (1998). Revisions to the algorithm, allowing it to handle a large number of observations by the sequential assimilation of observation batches, were described in Houtekamer and Mitchell (2001). The introduction of time interpolation, which resulted in a truly 4D algorithm, is described in Houtekamer and Mitchell (2005). We note that, in a perfect-model environment, the current algorithm can produce an ensemble spread that well represents the ensemble mean error without any need for covariance inflation (Houtekamer et al. 2009, Figs. 2 and 3).

a. Model–analysis communication

Data assimilation with the EnKF, as was the case with traditional operational intermittent data assimilation systems (Daley 1991, Fig. 1.11), consists of an alternating sequence of short-range forecasts and analyses. The short-range forecast is used as the prior estimate of the future state (sometimes also referred to as trial field or background). Once observations are available, they are combined with the prior in the analysis step to give an a posteriori estimate of the current state of the atmosphere. Subsequently, starting from the new estimate, we can perform the short-range forecast that will be combined with future observations when these become available. A specific aspect of the operational application is that there is some period during which we have to wait for future observations to arrive. We want this period to be relatively long to permit a timely recovery from any delay (perhaps caused by computer maintenance) in the operational procedure. Since we have to wait, it is natural to write the trial fields to files. Without this restriction, in a research and development or reanalysis application, one can consider configurations that avoid using files for the communication between the model and the analysis (Nerger and Hiller 2013). Of course, in an EnKF using 192 members, the number of files can become large and the management of these files requires special consideration.

In the Canadian EnKF, the Global Environmental Multiscale (GEM) model (Côté et al. 1998; Girard et al. 2014), which is used for the short-range forecasts, and the EnKF assimilation step use fairly distinct software packages. Using the same domain decomposition for both systems as in Keppenne and Rienecker (2002), if it

were judged desirable, would require a major development effort. Thus, currently the GEM model uses a certain domain decomposition for optimal parallel performance and the EnKF uses a different decomposition. Both algorithms have to swap between a standard global file format and a decomposed format for storage in memory. The associated transposition costs have been a source of concern (Lorenc 2003, section 4f).

b. Some key aspects of the EnKF

The main steps of an analysis algorithm can be viewed as (i) the computation of the innovations, given the background and the observations, and (ii)–(iv) the computation of the analysis increment on the analysis grid, given the innovations. In the Canadian EnKF, key features of the latter computation are (ii) the sequential algorithm, (iii) localization using a Schur product, and (iv) the solution of the analysis equations to yield the analysis increment.

1) CALCULATION OF INNOVATIONS USING TIME INTERPOLATION ALONG MODEL TRAJECTORIES

To permit time interpolation, the Canadian EnKF uses an extended state-vector approach (Anderson 2001; Houtekamer and Mitchell 2005); that is, the state vector is extended to include the ensemble trajectories mapped to observation locations using the forward operator $\mathcal{H}$. To permit the execution of $\mathcal{H}$, we need to read the ensemble of precomputed model trajectories. This is different from, notably, the 4D-variational method, in which the temporal component of $\mathcal{H}$ and its adjoint $\mathcal{H}^*$, are calculated during the minimization algorithm using tangent linear and adjoint versions of the forecast model (Courtier et al. 1994). Thus, in the EnKF, model trajectories can be precomputed but they must somehow be saved with sufficient precision. In a 4D-variational algorithm, one cannot precompute trajectories and therefore one must have a highly efficient linearized model and corresponding adjoint (Isaksen 2012).

2) THE SEQUENTIAL ALGORITHM

For the computation of the analysis increment, we can use either a sequential method (Houtekamer and Mitchell 2001) or a domain decomposition method like the local ensemble transform Kalman filter (LETKF; Szunyogh et al. 2008). Both of these methods exploit the fact that the impact of observations gradually tends to zero with increasing distance. In the sequential method, the analysis problem is subdivided into a number of manageable matrix problems by means of a separation of the observation file into a number of batches, which can be assimilated in sequential order. In the most extreme case, each observation batch consists of exactly one observation (Anderson 2001; Whitaker and Hamill 2002) and the number of passes through the sequential algorithm equals the number of observations. In the domain decomposition method, the analysis equations are solved for each grid point considering all observations that can have an impact. The potential performance of the LETKF on a system with $O(1000)$ processors is discussed by Szunyogh et al. (2008). Yang et al. (2009) have proposed interpolating analysis weights as a further improvement to LETKF efficiency.

In our system, we decided early to use a sequential method. In this context, it is possible to implement cross validation using subensembles (Mitchell and Houtekamer 2009). In cross validation, to assimilate observations into one set of trial fields, the Kalman gain matrix is computed using the trial field covariances obtained from the remaining members. Currently, at operations, we use a four-subensemble configuration. Thus, for each subensemble, which has one-quarter of the total number of members, the remaining three-quarters of the members are used to compute the Kalman gain. It follows that, in our current algorithm, a similar set of equations is to be solved 4 times. Note that these four different problems use different sets of three, of the four, subensembles and consequently, to avoid repeating calculations in computing covariances, we use partial sums over subensembles.

3) LOCALIZATION USING A SCHUR PRODUCT

To filter noisy estimates of low correlations at remote distances, we use a Schur product, denoted $\circ$, of ensemble-based covariances $\mathbf{P}$, with a localizing function ρ having finite support (Gaspari and Cohn 1999; Houtekamer and Mitchell 2001). In matrix form, this can be written as

$$\mathbf{P}_{\mathrm{loc}} = \rho \circ \mathbf{P}. \tag{1}$$

These matrices are dimensioned $N \times M$ and do not need to be square. To evaluate the Schur product, the matrices are multiplied element by element:

$$\mathbf{P}_{\mathrm{loc},ij} = \rho_{ij}\mathbf{P}_{ij}, \quad i = 1, \ldots, N, \quad j = 1, \ldots, M. \tag{2}$$

The scalar ρ_{ij} is obtained by evaluating a correlation function for the distance between elements i and j. We use the compactly supported fifth-order piecewise rational function given by Gaspari and Cohn [1999, Eq. (4.10)] for this purpose. Due to the finite support of the correlation function, the matrix $\mathbf{P}_{\mathrm{loc}}$ is sparse. As discussed in Houtekamer and Mitchell (2001), this sparseness can be used to define a number of independent regions on the globe where the analysis equations can be solved independently. With current localization parameters,

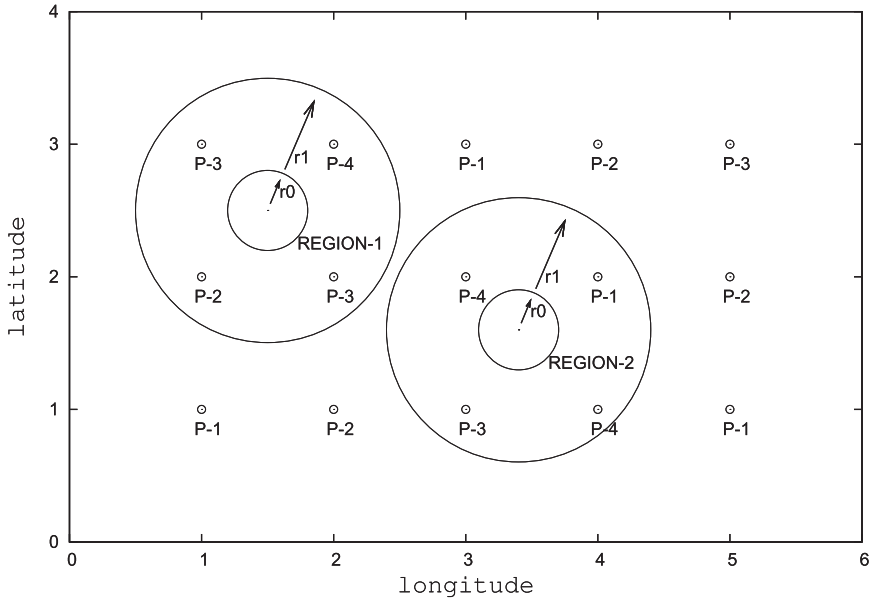

FIG. 1. Illustration of two parallelization strategies in a limited-area example with five longitudes, three latitudes, and two regions with observations. Strategy 1: Observations in each region of radius r0 are assimilated together in the same batch. The Cholesky decomposition (3) for the two regions can be done by different processes. Strategy 2: The 15 grid points are assigned in a round-robin manner to the four available processes (P-1 to P-4) to parallelize the solution of (6)–(8). Here, we only need to consider grid points out to a distance r1 from the edge of the circle with radius r0.

motivated by balance considerations (Mitchell et al. 2002), it is possible to define up to 13 independent regions on the globe and deal with them simultaneously during one pass of the sequential algorithm. (See Fig. 1 for an example where two independent regions in the domain have observations.)

4) SOLVING THE ANALYSIS EQUATIONS

Finally, following Evensen and van Leeuwen (1996), Cohn et al. (1998), and Houtekamer and Mitchell [2001, Eqs. (7) and (8)], the analysis equations are solved in observation space using a Cholesky decomposition (Golub and Van Loan 1996, section 4.2) and subsequently the solutions are transformed to the forecast-model space.

3. Space requirements

When running a software application on a computer system, one may be limited by a number of factors. In the case of an ensemble system, in which multiple realizations of variables are needed, the required disk and memory space may actually be limiting factors.

a. Disk storage requirements

For the configuration described above, with a 600 × 300 horizontal grid and 74 vertical levels and with information on four different 3D variables (two wind variables, temperature, and humidity), a model state consists of over 50 000 000 variables. Taking 4 bytes (B) per variable amounts to 200 MB for a model state. Using only 16 bits to cover the range between the extreme values for each horizontal field, this reduces to 100 MB. With a subsequent data compression, which has no further impact on precision, the observed file size reduces to approximately 57 MB. In the EnKF, to permit time interpolation, we use trial fields at every hour of a 6-h window. One trial field trajectory thus consists of seven individual states for a total of approximately 400 MB. It follows that an ensemble of 192 trial field trajectories weighs in at about 77 GB.

When running a longer data assimilation cycle, one wants to be able to rerun problematic cases. For instance, if a file turns out to be damaged, one wants to be able to return to a previous state that is still available. Thus, to actually run a data assimilation cycle with parameters as above on a POWER7 cluster, we need to have 250 GB of fast disk space on that cluster. To put this value into context, we note (D. Bouhemhem 2012, personal communication) that a total of 62 TB of fast disk space is available for research on each POWER7 cluster by $O(100)$ users. Thus, running one EnKF experiment will take a significant fraction of the disk-space resource available to a single user. This needs to be taken into

consideration when upgrading the resolution of the EnKF system.

At the minimum, when running an EnKF assimilation cycle for research purposes, we need to save fields with the ensemble mean and standard deviation as well as some diagnostics with respect to observations for a total of about 200 MB every 6 h of the cycle. This is a fairly modest amount considering also that we can use slow local disks for this purpose.

At the maximum, when saving the ensemble of trial field information for offline experiments with an ensemble–four-dimensional variational data assimilation (En-4D-Var; Buehner et al. 2010a,b) approach, intended for use by the global deterministic analysis and prediction system, we need to store 77 GB of information for every 6 h of experiments on disks of intermediate speed. If input is needed for a 4-month period (e.g., to prepare a proposal for operational implementation of the En-4D-Var), this amounts to a total of approximately 40 TB. It may be noted here that, in the early summer of 2012, two EnKF experiments running on the POWER7 could generate trial-field information at a substantial rate that exceeded the average write speed of the mass storage system (on tapes) at our center.

b. Memory requirements

On the first generations of computer systems, an efficient use of computer memory was one of the aspects of a good programming style. Due to increasing quantities of available memory, efficiently dealing with available memory has not been a critical issue during most of the development phase of the Canadian EnKF. Modern computer systems, however, have processes with relatively fast pure processing speeds and relatively slow memory access. To alleviate the problem, there is a sequence of memory caches between the main memory and the execution units. The POWER7, for instance, has a fast 32-KB "L1" data cache close to the execution units. Ideally, a number of numerical operations will be performed on variables present in the L1 cache before new variables are requested from the other slower caches and memory. A recent code optimization, relating to the combination of partial sums and the subsequent application of covariance localization, resulted in a more intensive use of the L1 cache contents and fewer accesses to the main computer memory. This reduced the execution time of a near-operational configuration by as much as 44%.

The current EnKF software system consists of a set of 15 individual programs, partly listed in appendix B, which use shared software libraries. When the number of model coordinates N_{model}, the number of observations N_{obs}, or the ensemble size N_{ens}, is increased, there will be some programs in the EnKF software system that will require increased array sizes. With continued increases, as tested in this paper, some processes will start exceeding available memory amounts (leading to a software failure). An obvious remedy, discussed in appendix A, is to assign more cores to each process. Parallelization within a process has recently been implemented using OpenMP directives. This permits using the processing power of the POWER7 with reasonable efficiency when the amount of memory per core is limited. In the EnKF experiments discussed in this paper, we use at most two OpenMP threads per MPI process.

4. Numerical components of a data assimilation cycle

In this section, we discuss the different steps in the algorithm in more detail. To guide the discussion, we will use timings of a configuration with the same parameters as in the EnKF delivered to the Canadian Meteorological Centre (CMC) in late 2012 and which became operational in mid-February 2013. This system features $N_{\text{ens}} = 192$ members. For the observations, we used data for the 6-h assimilation window centered on 0000 UTC 28 January 2011. The total number of observed values to be assimilated at this analysis time is $N_{\text{obs}} = 717\,411$. At a different analysis time, a somewhat different value of N_{obs} would be obtained, but it is hard to imagine that this would change results in a qualitative manner. The number of independent model coordinates is $N_{\text{model}} = 54\,000\,000$.

a. Preprocessing of observations

The EnKF benefits from many observation processing steps that are done in the context of the global deterministic prediction system of our center. This includes the selection of observations to be assimilated, the bias correction, the thinning of observations, some sanity checks, as well as a background check procedure. Thus, the EnKF system receives a fairly small and clean set of observations as input. These preprocessing steps, which are used in the deterministic system and from which we benefit, are beyond the scope of the current paper.

Historically, all programs currently used in the EnKF system (see appendix B for more details) were single-CPU (i.e., running as a single process using only one thread) programs. As the system evolved and expanded, some programs started to take a noticeable amount of time (exceeding, say, 1 min of wall-clock time) and became the subject of an optimization or parallelization effort. For some simple tasks not to be discussed here (including some of those mentioned in appendix B), no such analysis was ever performed.

TABLE 1. Processing times for some tasks that are executed prior to the computation of the analysis ensemble. These tasks are on the critical path toward user products. The workload is the product of the execution time and the number of cores used. Program labels are as in appendix B.

Task	Program	Time (s)	No. of cores used	Workload (s)	Time critical
Conversion of observation formats	B5	10	1	10	Yes
Determination of height of observations	B7	35	1	35	Yes
Application of the forward operator	B6	176	64	11 264	Yes
Quality control for satellite winds	B8	10	1	10	Yes
Sort for sequential processing	B9	13	1	13	Yes
Total		244	—	11 332	

The times required for different preprocessing steps of the EnKF are listed in Table 1. The dominating component is the evaluation of the forward operator $\mathcal{H}$. This operator is evaluated once for each of the N_{ens} members and for each of the N_{obs} available observations. Thus, the workload is $O(N_{\text{obs}} \times N_{\text{ens}})$. Since we use an extended state-vector approach (Anderson 2001), the operator can be evaluated outside of the main analysis program in a preprocessing step. To parallelize this step, we observe that the computation for different members is virtually independent. The only dependence between members is that, in our algorithm, the decision to accept or reject an observation must be shared between all members. Thus if, for instance, a radiance operator cannot be evaluated because the background values for one or more members are beyond certain bounds, the corresponding observation must be rejected for all members. In our current software system, each of 64 one-thread processes reads the members for which it has to perform the forward operators. This strategy might need to be revised if the number of available input/output (I/O) paths were substantially smaller than N_{ens}. Also, as grid sizes become larger, more memory will be required, which could necessitate using more cores per MPI process.

It would also be possible to evaluate the forward operator in the analysis program itself. As discussed in Szunyogh et al. (2008), this would lead to more complex code as one would have to also consider how information is distributed for the analysis. In addition, having the forward operator with time interpolation in the analysis program would necessitate having all time levels in the analysis state vector, which would result in a much longer state vector, as discussed by Houtekamer and Mitchell (2005, section 4e). The current extended part of the state vector has only N_{obs} elements.

The sorting program organizes observations into batches for sequential processing. In the original algorithm (Houtekamer and Mitchell 2001, Fig. 1), to sort N_{profile} observation profiles, a given profile could be considered many times for inclusion in a region. With increasing numbers of observations, this started to become expensive. In a revised formulation, observation profiles are first assigned to a predefined set of small areas [approximately $(1000\,\text{km})^2$] on the sphere. This can be done at minimal cost [$O(N_{\text{profile}})$] and allows groups of neighboring profiles to be identified much more quickly than before. We still impose a maximum $p_{\max} = O(1000)$ on the number of observations that can be assimilated in one batch for each independent region on the sphere. This value is limited to $O(1000)$ to keep the fraction of time to be spent obtaining the solution to the analysis equations in observation space [denoted $\mathbf{u}_i$ in (3) below] fairly small. In practice, since the introduction of vertical layers with different localization parameters (Houtekamer et al. 2014), the regions have limited vertical, as well as horizontal, extents. This has increased the total number of regions, resulting in fewer observations per region. Currently, regions generally contain 200–300 observations and the maximum $p_{\max} = 900$ is attained only in the most densely observed areas. The computational cost of the revised sorting algorithm is negligible for current observation volumes ($N_{\text{obs}} \approx 700\,000$).

b. Computation of analyses

Execution times in this section have been obtained using 192 members, a 600×300 horizontal grid, and 717 411 observations. These parameters correspond to the configuration that was transferred to operations in late 2012. While the configuration for operations uses 256 processes each using one thread, the EnKF code has been further optimized in the course of writing this paper. In this section, we present some results obtained with 288 processes each using one thread. This experiment is denoted ''refgrid'' in Table 2. With this configuration, the total wall-clock time required to obtain the final analysis ensemble is 675 s. All analysis experiments have been performed 4 times to reduce the estimation error in reported execution times.

The main parallelization strategy is to distribute the horizontal locations of the state vector (i.e., the model state and its extended part) over the available processes

TABLE 2. List of experiments. Each experiment is given an abbreviation for quick reference. The number of threads is the product of the number of MPI processes and the number of OpenMP threads per MPI process. The number of longitudes and latitudes specifies the horizontal grid. Here, N_{ens} is the ensemble size and N_{obs} is the number of observations. All configurations have 74 levels. The wall-clock time is the total elapsed time for the program that computes the analyses. These times have been obtained as an average over four realizations.

Expt	Threads	Grid	N_{ens}	N_{obs}	Wall-clock time (s)
str192	192 × 1	600 × 300	192	717 411	950
str384	384 × 1	600 × 300	192	717 411	541
str768	768 × 1	600 × 300	192	717 411	347
str1536	1536 × 1	600 × 300	192	717 411	263
str3072	3072 × 1	600 × 300	192	717 411	240
mem192	96 × 2	600 × 300	192	717 411	1119
mem256	128 × 2	600 × 300	256	717 411	985
mem384	192 × 2	600 × 300	384	717 411	949
mem512	256 × 2	600 × 300	512	717 411	936
mem768	384 × 2	600 × 300	768	717 411	1008
refgrid	288 × 1	600 × 300	192	717 411	675
biggrid	512 × 1	800 × 400	192	717 411	700
obs1	192 × 2	600 × 300	192	717 411	602
obs2	384 × 2	600 × 300	192	1 434 822	786
obs3	576 × 2	600 × 300	192	2 152 233	887
obs4	768 × 2	600 × 300	192	2 869 644	1010

in a round-robin manner. Thus, successive horizontal locations are assigned to successive processes. When no processes are left, the next horizontal location is assigned to the first process, after which the procedure continues. (See Fig. 1 for an example with 15 grid points and four processes.) This simple tiling strategy is possible because the forward operator $\mathcal{H}$, which includes a horizontal interpolation, has been precomputed as discussed in the previous subsection. Another possible strategy would be to assign horizontal locations randomly to processes (Anderson and Collins 2007). Currently, in our applications, the number of horizontal locations is $O(100\,000)$ whereas the number of processes is only $O(100)$. Thus, each process has to handle $O(1000)$ horizontal locations and the first processes will handle at most just one more horizontal location than the other processes.

On the other hand, information in observation space is typically shared between all processes. It is noted that, by virtue of the sequential algorithm that organizes observations into batches, vectors in observation space tend to be short with $O(1000)$ elements. The communication of these vectors accounts for a substantial nonscalable cost (Lorenc 2003, section 4f) with our current code, as we will see in section 4b(3) below.

In what follows, we discuss the main steps of the EnKF algorithm and their execution times, starting with the reading of the trial fields and their distribution over the available processes until the writing of the final analyses. We note that an overview of the main steps involved in the processing of a single batch was presented in Houtekamer and Mitchell (2001, section 4d). The adoption of an extended state-vector approach prompted two changes to that description: (i) step 2, the forward interpolation, is no longer part of the analysis itself but is part of the preprocessing, as already mentioned, and (ii) an additional step, to update the extended part of the state vector, is now required.

1) READING THE TRIAL FIELDS

On our current computer system, different processes can simultaneously read different files. The analysis program will read the trial fields in a round-robin manner with subsequent processes reading subsequent members. The fields are subsequently distributed over the processes as described above (i.e., vertical grid column by vertical grid column). This entire procedure is fast and takes only about 8 s of wall-clock time (Table 3), making the cost almost negligible.

By design, the entire set of meteorological fields corresponding to one particular ensemble member is read by one particular process. In the configuration denoted "str192" in Table 2, each of the $N_p = 192$ processes reads exactly one of the 192 ensemble members. When N_p becomes larger than N_{ens}, some processes remain idle and the procedure will no longer scale. A supercomputer may also have a limited number of active I/O channels preventing efficient simultaneous I/O for a large number of files. For instance, each of our POWER7 clusters uses five nodes for I/O operations. Therefore, in Table 3 (and later in Table 6), we mark the I/O procedures to be sequential.

Whereas we may be close to saturating the I/O system of our current computer, we expect future computers to have more I/O channels. It is thus quite possible that the file I/O will remain negligible on future systems.

2) USE OF SMALL, BUT FULL, WORK VECTORS

Each step of the sequential algorithm computes analysis increments for a limited number of small independent regions, as discussed in section 4a above (see also Fig. 1). To avoid the occurrence of large sparse matrices and to foster efficient use of the memory caches, each process preselects the background values at the affected grid points, from the full set of grid points it is responsible for, and puts them in a small vector. When the analysis increments for the regions have been computed, the process applies the opposite procedure to update the set of background values for which it is responsible. These two administrative procedures each take about 12 s on average to execute (not shown). Since with more

TABLE 3. Execution times (s) for main components of the analysis program in strong scaling experiments. These times have been obtained as an average over four realizations. A routine is considered sequential if its execution time does not decrease markedly with an increasing number of processes and is parallel otherwise. The total time spent in both types of routines is given. The total time (s) accounted for is only slightly lower than the total wall-clock time (s).

Description of routine	Type	str192 192 × 1	refgrid 288 × 1	str384 384 × 1	str768 768 × 1	str1536 1536 × 1	str3072 3072 × 1
Read trial fields	Sequential	8.58	8.14	9.26	9.69	9.05	8.09
Communicate observations	Sequential	29.84	28.17	26.40	21.58	19.35	20.99
Communicate $\mathcal{H}(\Psi)$	Sequential	14.46	19.74	24.63	40.46	60.37	84.89
Perturb observations	Sequential	13.15	13.15	13.15	13.15	13.15	13.18
Set up $[\rho \circ (\mathcal{H}\mathbf{P}\mathcal{H}^{\mathrm{T}}) + \mathbf{R}]$	Sequential	18.34	18.30	18.32	18.33	18.35	18.39
Solve for $\mathbf{u}$ in observation space	Sequential	2.60	2.60	2.58	2.58	2.60	2.60
Communicate $\mathbf{u}$	Sequential	4.63	4.99	5.22	6.13	6.98	9.15
Write analyses	Sequential	12.16	10.06	9.43	8.44	8.15	7.95
Total sequential	Sequential	103.76	105.16	108.99	120.34	138.00	165.25
Update state vector	Parallel	812.64	542.86	406.54	203.41	102.61	52.12
Update extended state	Parallel	16.73	10.04	8.25	4.16	2.47	1.62
Total parallel	Parallel	829.37	552.89	414.79	207.57	105.08	53.74
Total accounted	Any	933.13	658.05	523.78	327.92	243.08	218.99
Wall clock	Any	949.67	675.16	541.36	346.95	263.33	239.81

processes each individual process becomes responsible for fewer grid points, these procedures scale well.

3) COMMUNICATE FORWARD OPERATOR VALUES

When a new batch of observations is to be assimilated, each process extracts the corresponding values of the forward operator applied to the state vector, $\mathcal{H}(\Psi_i)$, for all ensemble members $i = 1, \ldots, N_{\mathrm{ens}}$, from its part of the extended state vector and broadcasts these values to all the other processes. Note that all processes will need these values to estimate the Kalman gain matrix as in (6) below.

For the refgrid configuration, this communication step takes 20 s but this time increases when more processes are used (Table 3). In appendix C, the increasing communication cost is investigated and a more efficient communication strategy is proposed.

4) PERTURBATION OF OBSERVATIONS

At each step of the sequential algorithm, the observations in the current batch are communicated to each process and then, using the same random number generator and seed, perturbed with random numbers. Since each of the processes perturbs all of the observations, this procedure does not scale and requires a time proportional to $N_{\mathrm{obs}} \times N_{\mathrm{ens}}$ independent of N_p. The perturbation of the observations with random numbers takes 13 s (Table 3). The rationale for this procedure, which dates back to our initial development of the sequential algorithm, is that "moving data between processors is slow compared to performing numerical operations upon data" (Houtekamer and Mitchell 2001, p. 130).

5) CHOLESKY DECOMPOSITION

A Cholesky decomposition is used to solve the equation

$$\mathbf{u}_i = [\rho \circ (\mathcal{H}\mathbf{P}\mathcal{H}^{\mathrm{T}}) + \mathbf{R}]^{-1}[\mathbf{d}_i - \mathcal{H}(\Psi_i)], \quad i = 1, \ldots, N_{\mathrm{ens}}, \tag{3}$$

where $\mathbf{R}$ is the observation-error covariance matrix, $\mathbf{d}_i$ is the ith (perturbed) observation vector, and $\mathcal{H}(\Psi_i)$ is the ith precomputed trial value equivalent. The solution vectors $\mathbf{u}_i$ are in observation space and have the same length, which equals the number of observations assimilated in the current region.

Before we can solve (3), we need to first compute the matrix $\mathcal{H}\mathbf{P}\mathcal{H}^{\mathrm{T}}$, multiply it by the localizing operator ρ, and add $\mathbf{R}$. Subsequently, in a second task, we can obtain the set of vectors $\mathbf{u}_i$, $i = 1, \ldots, N_{\mathrm{ens}}$, using a standard Cholesky decomposition and back-substitution procedure. Finally, in the third task, these solution vectors need to be communicated to all N_p processes.

At each step of the sequential algorithm, we have to solve (3) for each of four subensembles and for each of up to about 13 different regions. The first two tasks are performed in parallel in a round-robin manner for the about 50 sets of equations. We note that, in our current configuration, N_p tends to be substantially larger than 50 and thus many processes remain idle. The third task, the communication of the solution vectors, is a global operation that can be performed once all sets of equations have been solved. In total, with current parameters, the program spends only 26 s on these three tasks (Table 3).

To benefit from the idle processes and obtain better meteorological results, we plan to increase the number of subensembles from four to eight (Mitchell and Houtekamer 2009). This will both double the number of equations that need to be and can be solved simultaneously and increase the fraction of the ensemble used in the computation of the gain from 3/4 to 7/8.

It would be possible to parallelize the Cholesky decomposition (Benkner et al. 2002) by, for instance, using OpenMP. With our current modest matrix sizes and corresponding low cost (about 3 s; see Table 3) of solving (3), this is not useful. It could, however, be worthwhile in a context with larger observation counts and a larger number of observations per batch. Decreasing the number of steps through the sequential algorithm will normally have a positive impact on the quality of the analysis (Houtekamer and Mitchell 2001, Fig. 3).

6) UPDATING THE STATE VECTOR

A substantial 80% of the analysis time (i.e., 543 s) goes into solving the following equations (Houtekamer and Mitchell 2001):

$$\overline{\Psi^f} = \frac{1}{N_{\rm ens}} \sum_{i=1}^{N_{\rm ens}} \Psi_i^f, \tag{4}$$

$$\overline{\mathcal{H}(\Psi^f)} = \frac{1}{N_{\rm ens}} \sum_{i=1}^{N_{\rm ens}} \mathcal{H}(\Psi_i^f), \tag{5}$$

$$\mathbf{P}\mathcal{H}^{\rm T} = \frac{1}{N_{\rm ens}-1} \sum_{i=1}^{N_{\rm ens}} (\Psi_i^f - \overline{\Psi^f})[\mathcal{H}(\Psi_i^f) - \overline{\mathcal{H}(\Psi^f)}]^{\rm T}, \tag{6}$$

$$\mathbf{P}\mathcal{H}_{\rm loc}^{\rm T} = \rho \circ \mathbf{P}\mathcal{H}^{\rm T}, \quad \text{and} \tag{7}$$

$$\boldsymbol{\upsilon}_i = \mathbf{P}\mathcal{H}_{\rm loc}^{\rm T}\mathbf{u}_i, \quad i = 1, \ldots, N_{\rm ens}. \tag{8}$$

Here, to simplify the notation and the discussion, we neglect the use of subensembles for cross validation and the symbol Ψ only refers to the model variables (the extended part of the state vector is discussed in the next section).

The computation of the mean values in (4) and (5) is of negligible cost. In (6), we need to multiply the ensemble of model states by its corresponding values in observation space. Here, the differences between the $N_{\rm ens}$ state vectors and their mean are stored in a matrix of dimension $N_{\rm model} \times N_{\rm ens}$. The corresponding transposed result in observation space is stored in a matrix of dimension $N_{\rm ens} \times N_{\rm obs}$. For the matrix multiplication, we need to perform $O(N_{\rm model} \times N_{\rm obs} \times N_{\rm ens})$ operations. Fortunately, here we can already exploit the sparseness that will be introduced by the localizing operator in (7), to evaluate (6) only for the grid points that can be affected by the current set of observations. The resulting matrix $\mathbf{P}\mathcal{H}^{\rm T}$ is of dimension $N_{\rm model} \times N_{\rm obs}$. In (7), this matrix is multiplied elementwise, as in (2), by a localizing matrix, which is an operation of cost $O(N_{\rm model} \times N_{\rm obs})$. The matrix of vectors $\mathbf{u}_i$ is of dimension $N_{\rm obs} \times N_{\rm ens}$ and, consequently, the multiplication in (8) also requires $O(N_{\rm model} \times N_{\rm obs} \times N_{\rm ens})$ operations. In this discussion, for convenience, we neglected the role of the sequential algorithm, which helps to make matrices of manageable size but has no impact on the total operation count for (4)–(8).

If it were not for the localizing step (7), these matrix operations could be performed in a different order [Mandel (2006), Eq. (4.1)] to arrive at a more advantageous operation count.

We note that since the model state has been distributed over the processes (Fig. 1), (4)–(8) parallelize well (Table 3). This, combined with the use of highly optimized matrix multiplication routines provided with the computer, makes the computation of the analysis increments $\boldsymbol{\upsilon}_i$ with the above method feasible.

7) UPDATING THE EXTENDED PART OF THE STATE VECTOR

To update the extended part of the state vector, we solve the set of equations, (4)–(8), with Ψ_i^f in (4) and (6) replaced by the elements of the extended part of the state vector. This part of the state vector is of length $N_{\rm obs}$ as compared to the "regular" part of the state vector, which corresponds to actual model coordinates, which is of length $N_{\rm model}$. Typically, for NWP assimilation problems, we have $N_{\rm obs} \ll N_{\rm model}$. About 10 s are used to update the extended part of the state vector.

Similar to the discussion about updating the state vector, we obtain terms with operation counts of order $O(N_{\rm obs} \times N_{\rm obs} \times N_{\rm ens})$ and $O(N_{\rm obs} \times N_{\rm obs})$. As we will see, these terms become relatively more costly when only $N_{\rm obs}$ is increased.

8) WRITING THE ANALYSIS

The analysis program writes the analysis fields in a round-robin manner with each process writing the analysis of one member until no analyses are left. This takes a fairly negligible 10 s. Some of this time is used to collect the analyses from the different processes, prior to writing the fields to disk.

We note that earlier versions of our code did not assume it possible to simultaneously write many different files and used only a small set of processes to write to disk. At that time, the collection of analyses was also done by a very small number of processes and the usage of the memory cache was not optimal. With increasing problem

TABLE 4. Processing times for some of the tasks that are executed, in a continuous data assimilation cycle, after the computation of the analysis ensemble. The workload is the product of the execution time and the number of cores used. Program labels are as in appendix B. One task uses no programs from our library.

Task	Program	Time (s)	No. of cores	Workload (s)	Time critical?
Compute analysis mean and std dev	B1	80	8	640	Yes
Add perturbations, compute statistics	B1, B3, B4	235	32	7520	No
Add surface fields for the model	—	180	16	2880	No
Sample 20 initial conditions	B12	146	1	146	Yes
Generate input for database	B11	62	1	62	No
Precompute perturbation fields	B2	600	8	4800	No

size, we had to adjust this strategy. As with reading the trial fields, the use of simultaneous I/O has rendered the transposition costs (Lorenc 2003, section 4f) negligible.

c. *Postprocessing steps*

Several postprocessing steps are performed after the analysis. Those postprocessing steps, which are necessary to continue a data assimilation cycle with a late data-cutoff time, are listed in Table 4 together with their execution times. These modest times are mostly not of critical importance because they are not on the critical path that leads to 20 medium-range forecasts for the users of the global ensemble prediction system. Some steps on the critical path would benefit from optimization but they are not specific to data assimilation with the EnKF. We will not further consider any postprocessing steps in this paper.

d. *Integration of the model*

In Table 5, we present times associated with producing an ensemble of 9-h forecasts using the dynamical model. The N_{ens} ensemble forecasts can be integrated independently. The workload for the main task is 1 336 320 s which is about 7 times more expensive than obtaining the ensemble of analyses with 288 cores in 675 s (experiment refgrid in Table 2).

Our operational division has high priority on the computer and can perform all N_{ens} forecasts simultaneously using a total of 3072 cores (16 cores per ensemble member). With this amount of computer resources, the result is available within 10 min. This is a short time compared to the 6-h interval of real time between subsequent analyses. This difference makes for a robust system that can easily recover and catch up after a disruptive situation.

As currently designed, running an ensemble of N_{ens} integrations requires submitting $6N_{ens}$ tasks (the five tasks in Table 5, as well as a clean-up task) to the queuing system, which makes for a very visible system with many associated files. This procedure, however, is very robust in the sense that any problematic completion of a task can be conveniently investigated (since one knows which member had a problem with which task) and corrected (since only the affected task needs to be reexecuted).

Since, as we have seen, the workload for the integration of the model is larger than other workloads in the EnKF, we can use it to estimate the parameters of the biggest possible EnKF configuration that could feasibly be run at CMC operations with our current computers. Suppose we have all 8000 cores of one POWER7 cluster available for 4 h to integrate one ensemble of forecasts. Taking 16 cores for, say, 585 s to integrate one member (Table 5), we find that we could integrate over 12 000 members at the current resolution. Alternatively, keeping the number of members fixed at 192, we could increase the cost per integration by

TABLE 5. Execution times for tasks involved in producing an ensemble of short-range (9 h) integrations with the dynamical forecast model. Results are given for configurations "refgrid" and "biggrid," which, respectively, have 67- and 50-km resolution at the equator. The workload, which is given for the 67-km configuration, is the product of the execution time and the number of cores used.

Task	Time (s)		No. of cores per member	No. of cores	Workload (s)
	67 km	50 km			67 km
Create links for input files	60	33	1	192	11 520
Run the model entry program	30	32	1	192	5760
Integrate the model	435	927	16	3072	1 336 320
Reassemble model output	45	67	1	192	8640
Transfer to destination	15	50	1	192	2880
Total	585	1109	—	—	1 365 120

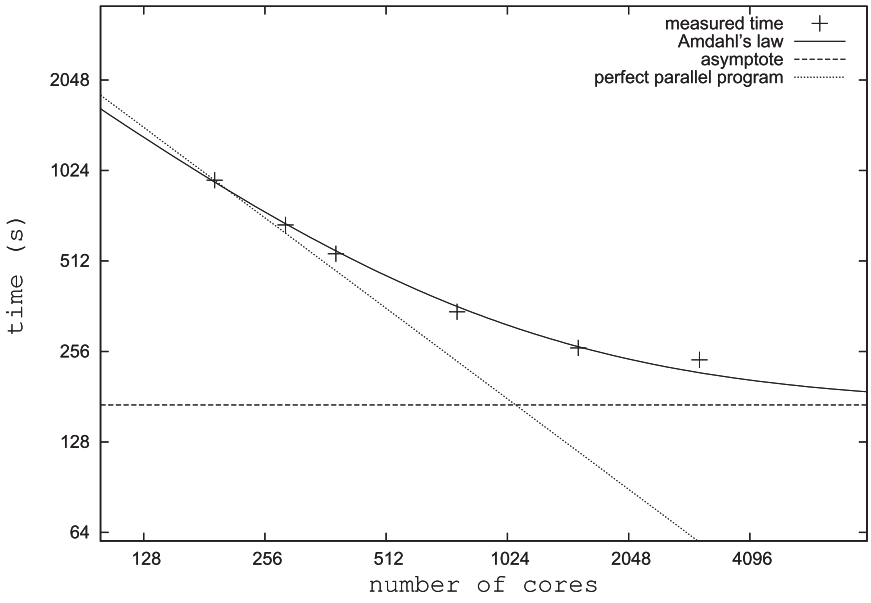

FIG. 2. Execution time is given as a function of the number of cores for a strong scaling experiment. The wall-clock time has been measured for configurations with 192, 288, 384, 768, 1536, and 3072 processes, each using only one core. The curve for Amdahl's law uses a parallel fraction of 0.998 85. The hypothetical "perfect parallel program" matches the measured time for 192 processes. Amdahl's law asymptotes to an execution time of 170 s. We cannot use more than 8000 cores on the POWER7.

a factor of 62.5. Of course, it would require a highly scalable assimilation system to deal with such a fascinating problem.

5. Scaling

In this section, we will consider two types of scaling. Strong scaling measures how the solution time varies with the number of processes N_p for a fixed total problem size. In weak scaling, the solution time is measured as a function of N_p when the problem size per process is kept fixed. In general, strong scaling is difficult to achieve because of Amdahl's law (Amdahl 1967). In practice, though, one may not even be interested in using a very large number of processes to obtain a result in an extremely short time. At an operational center, once the execution time of the analysis drops below, say, 15 min, there may be little need to try to achieve even faster execution. At that stage, one is likely to attempt to solve a bigger problem with a larger number of processes to obtain a higher quality analysis in the same amount of time.

In an EnKF, the computational cost may not be clearly dominated by a single term (e.g., Tippett et al. 2003, Table 1). When several terms are involved, the total cost becomes a function of the relative efficiency of implementation of the different terms. Based on our previous experience (Houtekamer and Mitchell 2001, p. 133), we will assume that updating the state vector dominates the cost [(6) and (8)]. As discussed in section 4 above, the cost of these operations has a term of order $O(N_{\text{model}} \times N_{\text{obs}} \times N_{\text{ens}})$ and for simplicity, when investigating weak scaling, we first assume that the analysis cost is proportional to the three parameters N_{obs}, N_{ens}, and N_{model}.

To obtain high quality meteorological results, increasing N_{obs}, N_{ens}, and N_{model} are all desirable and it is thus of some importance for us to understand more precisely how the computational cost behaves as a function of these parameters.

The experiments for this section are summarized in Table 2. In this section, we will try to understand the wall-clock times shown.

a. Strong scaling

For our analysis of strong scaling, we solve our standard meteorological problem with $N_{\text{ens}} = 192$, $N_{\text{obs}} = 717\,411$, and a 600×300 horizontal grid using different numbers of processes. We use 192, 288, 384, 768, 1536, and 3072 processes, with each process having only one thread. In Table 2, these experiments are labeled as str192, refgrid, str384, str768, str1536, and str3072, respectively. As seen in Fig. 2 and Table 2, the wall-clock time goes down from 950 to 240 s as N_p increases from 192 to 3072.

According to Amdahl's law, given a fraction P of the program that can be made parallel, the speed-up that can be obtained with N_p processes is given by

$$S(N_p) = \frac{1}{(1-P) + \frac{P}{N_p}}. \quad (9)$$

From the execution times in Table 2, we can estimate the parallel fraction to be approximately $P = 0.998\,85$.

In the first reported results with our algorithm (Houtekamer and Mitchell 2001, Table 2), we obtained a speed-up of 12.28 with 16 processes for a parallel fraction of $P = 0.9798$. Unfortunately, it is difficult to compare these P values because they derive not only from the evolving EnKF algorithm, but also from the different computer systems that were used (Hager and Wellein 2011, section 5.3.5). In the current complex computational environment, it is hard to envision a total number of instructions of which a certain fraction has been perfectly parallelized and of which the remaining fraction is entirely sequential. The main value of Amdahl's law is that it provides a reasonable model framework for interpreting experimental speed-up values (see Fig. 2).

The configuration with $N_p = 192$ features a speed-up of $S(192) = 157$ with respect to a hypothetical configuration with $N_p = 1$ that would complete in 41 h of wall-clock time. In practice, even with $N_p = 96$ and one thread per process, we do not obtain enough memory to solve our standard problem. In the other unrealizable extreme, taking the limit $N_p \to \infty$ in (9), we obtain an execution time of 170 s.

To better appreciate the nature of the sequential component of the EnKF, we have measured the time spent in different routines of the EnKF (Table 3). When we denote a routine as sequential, this means that, once 192 or more processes are used, the routine no longer behaves as if it were parallelized. For instance, the 192 trial fields are all read simultaneously provided we have at least 192 processes. Having more processes, we cannot read these fields more quickly and the routine thus appears to be sequential from the results in Table 3.

In agreement with the notion of having a parallel and a sequential component, we note an almost perfect speed-up for the routines that update the state vector and its extended part. The total time spent in the other routines is below 3 min, which is close to the estimate we had obtained from Amdahl's law. As currently implemented, the routine used to communicate the results of the forward operator takes more time when more processes are used (appendix C). This fact, which is also reflected in the poor fit to the 240 s measured with $N_p = 3072$ (see Fig. 2), makes us think that we cannot obtain our results substantially faster by increasing N_p beyond 3072 with our current algorithm.[2]

b. Weak scaling: Number of ensemble members

The weak scaling with respect to ensemble size is investigated with experiments mem192, mem256, mem384, mem512, and mem768, which use ensemble sizes of $N_{\text{ens}} = 192$, 256, 384, 512, and 768, respectively (see Table 2). Note that, in order to obtain sufficient memory for the experiment mem768, we use two threads per process for this set of experiments. The experiments str192 and mem192 both use 192 threads to solve the same meteorological problem. A comparison of their wall-clock times in Table 2 quantifies the advantage of using just one thread per process. For the also comparable configurations str384 and obs1, we again observe better performance when using just one thread per process.

We note in Table 2 and Fig. 3 that the actual execution time goes down, at least initially, as the ensemble size is increased. To better understand this result, we show the execution times for individual routines in Table 6. Here, in view of the relatively weak dependence of timings on N_{ens}, we also provide an estimated standard deviation. For the routine communicating observations we tend to obtain a relatively large range of timing results from the four realizations with identical configurations. This may be related to simultaneous activity by other users of the POWER7 cluster. For routines doing pure numerical work, like perturbing observations, we obtain more stable results.

We see that the amount of time spent in the parallel routines initially goes down substantially when N_{ens} and N_p are increased simultaneously, as is the case in these weak scaling experiments. Further analysis showed that a substantial amount of time is spent in operating upon partial sums and in localizing covariances. As mentioned in section 2b(2), the partial sums over subensembles were introduced to avoid repeating calculations due to our use of cross validation. Subsequent operations upon these partial sums no longer depend on N_{ens} and consequently their cost is only of order $O(N_{\text{model}} \times N_{\text{obs}})$. For each of the $N_{\text{model}} \times N_{\text{obs}}$ elements of the matrix $\mathbf{P}\mathcal{H}^{\mathrm{T}}$, we need to compute a localizing factor in (7). The above two operations do not depend on N_{ens} and have good strong scaling. For these weak scaling experiments, we can thus, analogous to Amdahl's law, model the time t_{par} spent in the parallel code as

[2] With the optimized distribution strategy described in the last paragraph of appendix C, we obtained an execution time of 175 s with $N_p = 3072$.

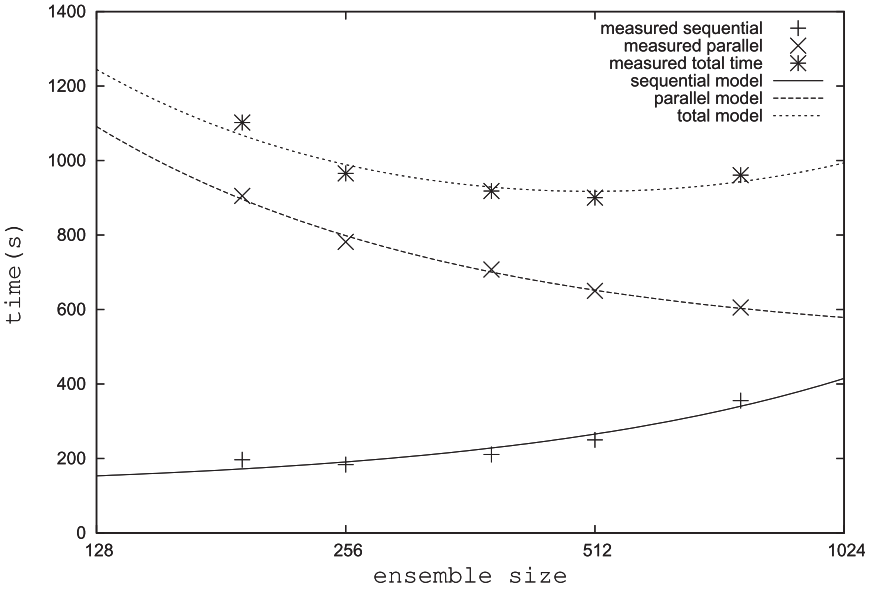

FIG. 3. Execution time is given as a function of the ensemble size for a weak scaling experiment. All configurations use two threads per process and the total number of threads equals the ensemble size. Routines have been classified as either parallel or sequential. Amdahl's law is used to model the parallel components, some of which show strong scaling since their cost does not depend on ensemble size. The model for the sequential component is linear in ensemble size. The model for the total execution time is the sum of the parallel and sequential models.

$$t_{\text{par}} = c_1 + \frac{c_2}{N_{\text{ens}}}, \tag{10}$$

where c_1 and c_2 are tunable constants, with the c_1 term corresponding to (6) and (8) and the c_2/N_{ens} term corresponding to the operations on the partial sums and (7). Note that in this weak scaling experiment, N_{ens} is proportional to N_p.

Another observation to make from Table 6 is that the time spent on some sequential routines increases substantially when N_{ens} and N_p are increased simultaneously. In part this reflects that the amount of work in these routines sometimes, such as when we need to perturb the observations [section 4b(4)], scales with N_{ens}. For other actions, such as when writing the N_{ens} analyses, one could have hoped for more constant execution times. Perhaps, here, the computer system is not able to efficiently write such large numbers of files in parallel. Consequently, we postulate the following model for the time t_{seq} spent in the sequential routines:

$$t_{\text{seq}} = c_3 + c_4 N_{\text{ens}}, \tag{11}$$

where c_3 and c_4 are tunable constants.

In Fig. 3, we see to what extent these two simple models fit the data from the five experiments. In view of the overall weak dependence of the total wall-clock time on problem size, we conclude that weak scaling is respected quite well for ensemble sizes of up to $O(1000)$ members.

c. Weak scaling: Number of grid points

The scaling with the number of grid points has been investigated with experiments refgrid and biggrid. Here, we go from a 600×300 to an 800×400 horizontal grid with a proportional increase in N_p (i.e., from 288 to 512). The small increase in wall-clock time (from 675 to 700 s; see Table 2) can be partly explained by the additional time required to read and write the bigger grids (not shown). Otherwise, the sequential routines do not clearly depend on N_{model} and the parallel routines mostly have a cost proportional to N_{model}. Thus, here, the weak scaling experiment is successful in that the computational cost is indeed roughly proportional to N_{model}. Consequently, due to the parallelization, simultaneously increasing N_{model} and N_p virtually leaves the wall-clock time unchanged.

As discussed in section 3b, an issue related to increasing N_{model} is that some programs start requesting more memory than is available per process. It would consequently require some coding effort, not necessarily even including the analysis program of which the scaling

TABLE 6. Execution times (s) for weak scaling with respect to N_{ens}. All five experiments use two threads per process. For each experiment, the total number of threads equals the ensemble size. The routines marked as "parallel" showed good strong scaling in Table 3. The standard deviation is the RMS of the standard deviations estimated from the four realizations for each of the five experiments.

Description of routine	Type	mem192 96 × 2	mem256 128 × 2	mem384 192 × 2	mem512 256 × 2	mem768 384 × 2	Std dev
Read trial fields	Sequential	22.15	25.45	26.00	32.66	47.19	3.58
Communicate observations	Sequential	105.01	74.76	71.27	73.81	98.05	43.96
Communicate $\mathcal{H}(\Psi)$	Sequential	9.81	13.21	21.11	30.30	54.01	2.10
Perturb observations	Sequential	13.15	17.53	26.26	35.00	52.45	0.02
Set up $[\rho \circ (\mathcal{H}\mathbf{P}\mathcal{H}^{\mathrm{T}}) + \mathbf{R}]$	Sequential	16.34	20.26	27.91	35.52	50.82	0.15
Solve for $\mathbf{u}$ in obs space	Sequential	2.62	2.82	3.17	3.53	4.25	0.01
Communicate $\mathbf{u}$	Sequential	4.20	5.43	7.65	10.02	14.91	0.65
Write analyses	Sequential	23.49	24.22	27.28	29.26	33.72	1.33
Total sequential	Sequential	196.77	183.67	210.66	250.11	355.39	44.70
Update state vector	Parallel	886.41	765.85	694.12	639.01	595.01	25.34
Update extended state	Parallel	18.76	15.76	13.29	11.14	10.38	0.08
Total parallel	Parallel	905.17	781.61	707.41	650.16	605.39	25.39
Total accounted	Any	1101.94	965.28	918.07	900.26	960.78	46.14
Wall clock	Any	1118.62	984.68	948.52	936.10	1007.70	45.74

is currently being discussed, to include an experiment with a 1200 × 600 horizontal grid.

d. Weak scaling: Number of observations

Our refgrid configuration uses the N_{obs} observations that were actually available to us for the analysis valid at 0000 UTC 28 January 2011. We have no straightforward way of obtaining a larger number of actual observations valid at the same time. To nevertheless perform scaling experiments, the same data have been repeated a number of times. For the weak scaling analysis, we compare the experiments obs1, obs2, obs3, and obs4 (see Table 2). Because of memory requirements of the bigger experiments, we used two threads per process for this sequence of experiments.

We see from the wall-clock times in Table 2 and Fig. 4 that the weak scaling with N_{obs} is fairly poor. Thus, using 4 times the number of processes only partially compensates for having 4 times the number of observations; in fact, experiment obs4 takes 1.7 times as long as experiment obs1 (Table 2). The delay can be explained largely by considering the seven routines listed in Table 7. For the communication routines it is natural that they should take more time when both more observations and more processes are used. With regard to perturbing the observations, it has been mentioned in section 4b(4) that this has not been parallelized and we see that the time here is simply proportional to N_{obs}. For the Cholesky decomposition, once N_{obs} is large enough that the matrices have their maximum size p_{max}, the time is expected to be proportional to N_{obs}. For the update of the extended state vector, we had noted an almost perfect scaling in the strong scaling experiment (Table 3). This term, however, scales as $O(N_{obs} \times N_{obs} \times N_{ens})$. Note that if N_{obs} and N_{model} were increased simultaneously by the same factor, the cost for the extended and regular parts of the state vector would increase by this same factor.

The wall-clock times are observed to depend linearly on the number of observation sets (Fig. 4). During the development of our solution algorithm and our parallelization method, we have always had to deal with parameter sets with $N_{model} \gg N_{obs}$ (e.g., the configuration refgrid has $N_{obs} = 717\,411$ and $N_{model} = 54\,000\,000$). Notably, we implemented an extended state vector method and it is the grid points that are evenly distributed over the processes (Fig. 1). While the routines listed in Table 7 can likely be further optimized or parallelized, for substantially larger observation counts, a different algorithm may have to be adopted. At this stage, it is not clear if variational solution algorithms can be efficient in the context of large ensembles. Pending an algorithmic breakthrough, accounting for observation error correlations could permit more information to be extracted from a given number of observations.

6. Discussion

The parallel algorithm used in the Canadian global EnKF has been presented. Some scaling experiments have been performed to measure the cost as a function of increasing ensemble size, model resolution, and the number of observations.

The total cost is currently dominated by the cost of generating an ensemble of trial field trajectories with the

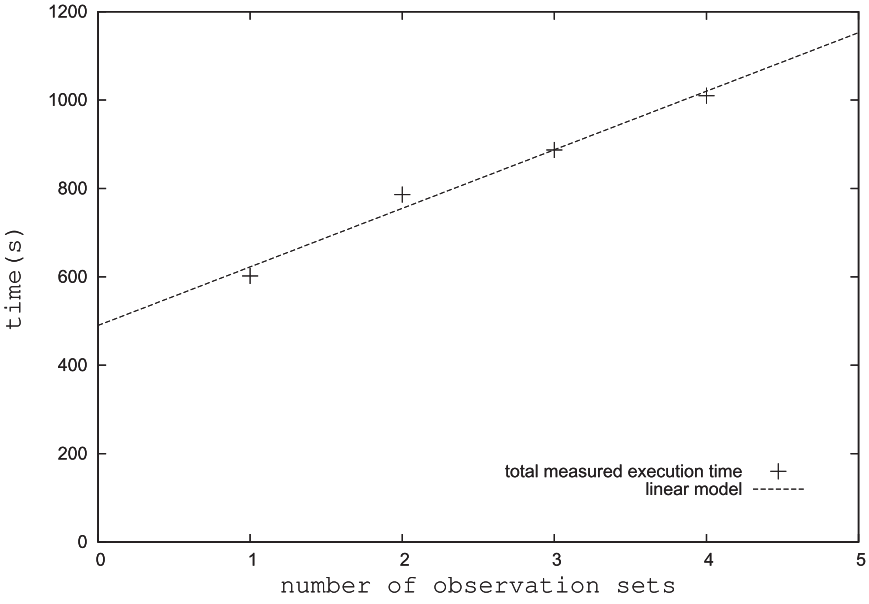

FIG. 4. Execution time is given as a function of the number of assimilated observation sets for a weak scaling experiment. All configurations use 192 processes per observation set and each process uses two threads. The model for the execution time is linear in the number of observation sets.

forecast model. However, this step is not time critical and it parallelizes almost perfectly. Thus, with respect to the forecast component of the EnKF, a substantially more costly configuration could in principle be run on our computer. Consequently, in this paper the focus is on the time-critical analysis component.

Some aspects of our analysis algorithm date back to our original research and development configuration (Houtekamer and Mitchell 1998), which used a 64×32 horizontal grid, three vertical levels, and approximately 1100 simulated observations. Over the last 15 yr, the EnKF algorithm has gradually evolved to permit more challenging parameters of the meteorological problem and to benefit from the improvements in the available computational platforms. The most significant algorithmic changes were the move to a sequential method with localization performed using a Schur product (Houtekamer and Mitchell 2001) and the introduction of an extended state vector (Houtekamer and Mitchell 2005). Two recent optimizations, which led to a substantial gain in performance, involved (i) a better use of the memory caches in the update of the state vector

TABLE 7. Detail for weak scaling experiments with respect to N_{obs}. A total of 192×2 threads are used per observation set. Of the 10 routines listed in Table 3, only the routines with substantially increasing times are listed here. The one ''parallel'' routine showed good strong scaling (Table 3) but now has poor weak scaling with respect to N_{obs}. The accounted delay is the sum of the delays for the listed routines with respect to obs1. The observed total delay is computed with respect to the wall-clock time. All reported times are an average over four realizations.

Description of routine	Type	obs1 192×2	obs2 384×2	obs3 576×2	obs4 768×2
Communicate observations	Sequential	48.73	125.92	110.68	147.37
Communicate $\mathcal{H}(\Psi)$	Sequential	15.61	47.92	94.45	149.90
Perturb observations	Sequential	13.16	26.31	39.43	52.55
Set up $[\rho \circ (\mathcal{H}\mathbf{P}\mathcal{H}^{\mathrm{T}}) + \mathbf{R}]$	Sequential	16.35	34.61	52.56	68.51
Solve for $\mathbf{u}$ in obs. space	Sequential	2.61	6.18	9.82	12.81
Communicate $\mathbf{u}$	Sequential	4.64	8.49	11.84	15.73
Update extended state	Parallel	10.63	19.36	28.43	38.80
Total measured	Any	111.73	268.79	347.21	485.67
Accounted delay	Any	0.00	157.06	235.48	373.94
Wall clock	Any	601.96	786.03	886.89	1009.63
Observed total delay	Any	0.00	184.07	284.93	407.67

[(4)–(8)] and (ii) a move to more parallel file I/O. Moving to future configurations, with $O(1000)$ threads, there is a need to optimize communications. In this context, we have recently begun investigating some troubling aspects of the timings for the communication of observations noted in Table 6 (large standard deviation of timings) and Table 7 (nonmonotonic increase of execution times). Preliminary results indicate that a substantial fraction of the time that we have ascribed to communication of observations is, in fact, due to the synchronization of all processes that is required before the actual communication can begin.

Excellent weak scaling results were obtained when changing either the ensemble size N_{ens} or the number of model coordinates N_{model}. To take advantage of this, a configuration with $N_{\text{ens}} = 256$ members configured in eight subensembles of 32 members (up from 192 members configured in four subensembles of 48 members) and with a 800×400 horizontal grid (currently 600×300) is being prepared for delivery to CMC operations in 2014. We saw that the current EnKF algorithm could support an increase by several multiples in the number of observations. Increases beyond that could be problematic and several options for dealing with this are currently being examined. These include (i) the more aggressive use of optimized library routines,[3] (ii) the optimization of communications, (iii) accounting for observation error correlations, and (iv) the use of a variational solution of the equations.

The computations involving the matrix $\mathbf{P}\mathcal{H}^{\mathrm{T}}$ scale as $N_{\text{model}} \times N_{\text{ens}} \times N_{\text{obs}}$ and dominate the analysis cost. Other important cost terms, for instance related to disk storage, scale as the product of only two of these parameters. Thus, the historical spectacular improvement of floating-point operations per second (FLOP) rates and the smaller improvement in, for instance, disk-storage capacity are a natural good match to the changing requirements of more ambitious EnKF configurations. Likely, future clusters at our center will have access to more compute cores and, to make optimal use of these, we intend to optimize the hybrid use of MPI and OpenMP. Unfortunately, at this stage, we do not know the specifications of our future supercomputers.

The ensemble square root filter (EnSRF), as implemented at the National Centers for Environmental Prediction (NCEP; Whitaker et al. 2008) to serve in a hybrid analysis system, is conceptually similar to our EnKF system and may qualitatively respond similarly to changing computational environments and problem parameters. Note, however, that the NCEP system does not use subensembles and, since it assimilates observations one at a time, does not employ the Cholesky decomposition procedure. The local ensemble transform Kalman filter (LETKF) is conceptually more different but is expected to scale well on thousands of processors (Szunyogh et al. 2008). It would be of interest if other groups would describe how they have addressed the various problems that were discussed in this paper and show their corresponding scaling results.

With regard to the En-4D-Var system, we saw, in section 3a, that storing all the trial field trajectories needed for an extended run of that system required a considerable amount of storage space. For current high-resolution experiments toward the 2014 operational delivery, we recently obtained access to an additional 120 TB of disk space. To alleviate the storage requirements and permit active two-way coupling, we are developing the infrastructure to simultaneously run a coupled En-4D-Var and EnKF system. Here, the En-4D-Var provides bias-corrected and quality-controlled observations to the EnKF, which in return provides flow-dependent background error statistics.

With regard to the EnKF itself, its computational evolution in the more distant future will depend not only on the specifications of our future computers, but also on how the algorithm itself evolves scientifically. As the parameter N_{model} continues to increase, the range of spatial scales that needs to be appropriately handled by the EnKF grows. These scales, ranging from global to the smallest spatial scales resolved by the model, and their associated dynamical balances should be respected in an efficient algorithm. If we want to continue to be able to use a limited number of ensemble members, some sort of multiscale approach for the localization, perhaps along the lines suggested by Zhang et al. (2009), will be required. Reducing the length of the assimilation window from the current 6 h could also help; it would, for example, reduce the number of observations to be assimilated at a given analysis time and reduce the distance the error covariances move over the time window. However, it is not obvious how to impose balance in such a system (Houtekamer et al. 2014) and reducing the length of the assimilation window would exacerbate spin-up problems. The parameter N_{obs}, too, has been increasing rapidly in recent years, mainly due to the ever larger amounts of satellite radiance data available and relaxed data-thinning parameters. As data density increases, the need to account for observation-error correlations becomes more pressing. Progress on emerging

[3] In the final stages of the review procedure for this paper, we found that we could reduce the cost of computing the matrix $\mathcal{H}\mathbf{P}\mathcal{H}^{\mathrm{T}}$, which is used in the expression $[\rho \circ (\mathcal{H}\mathbf{P}\mathcal{H}^{\mathrm{T}}) + \mathbf{R}]$, by approximately a factor of 10 by using an optimized matrix multiplication routine.

fundamental issues, such as these, will normally impact the formulation of the analysis algorithm.

Acknowledgments. We thank Djamel Bouhemhem for the generous allocation of disk space. We thank Ervig Lapalme, Laurent Chardon, and Michel Valin for their help in creating an EnKF benchmark for a future upgrade of the Environment Canada supercomputer. That exercise resulted in important optimizations of the EnKF code. We thank Michel Desgagné for his internal review of the paper as well as for numerous illuminating comments about the POWER7 architecture. We thank the anonymous reviewers for many valuable suggestions that improved and broadened the scope of the paper and pushed us to do the analysis leading to the optimization proposed in appendix C.

APPENDIX A

Computer Architecture

Processes communicate with each other by using message passing interface (MPI) routines (Gropp et al. 1994) that form a library that can be called from standard FORTRAN code. OpenMP (Chandra et al. 2001) is used to distribute work within a process to a number of threads. The hybrid use of MPI and OpenMP is discussed in depth in Hager and Wellein (2011, chapter 11).

To illustrate, Fig. A1 shows various ways of organizing a parallel program using the MPI and OpenMP protocols. In reality, each of our center's POWER7 clusters has approximately 250 nodes and each node has 32 cores. However here, for simplicity, we show an example with only two nodes, each having only four cores (Fig. A1a).

The simplest way of using this entire parallel computer is to run one process on each of the cores and have the processes communicate with each other by using MPI (Fig. A1b). It may happen, for instance because a process needs to store an entire meteorological state, that a core does not physically have the memory requested by an application. The simplest way to fix this problem is to assign the cores so that each process obtains two cores and the corresponding share of physical memory of the node (Fig. A1c). Since now only four, out of a possible eight, threads can run on the small computer, this simple solution is not likely to make efficient use of the computational resources. If even more memory per process is needed, one could run one process per node (Fig. A1d). It is not possible for a process to run on more than one node. When this limit is exceeded, one needs to redesign the software application.

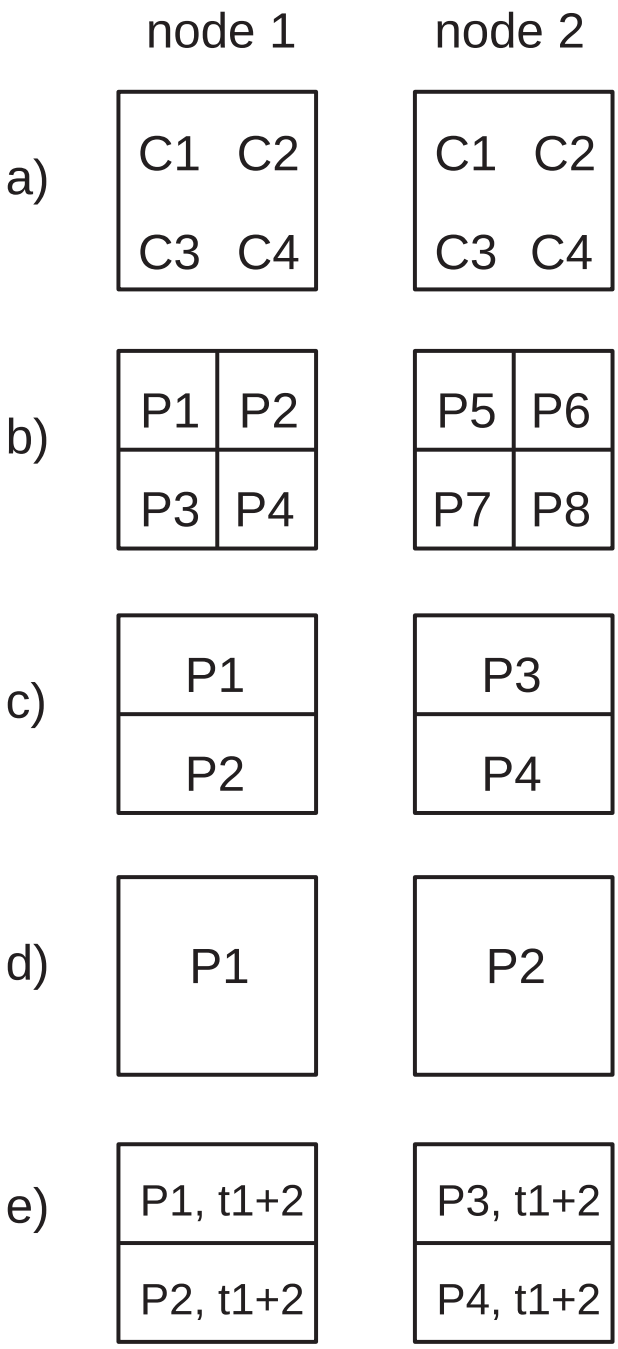

FIG. A1. (a) A small cluster consisting of two nodes. Each node has four cores that can share memory. (b) MPI is used to run eight processes that can send messages to each other. Under MPI, the processes cannot directly share memory. (c) Four processes are submitted so that each of the processes has twice the amount of memory as in (b). (d) Each of two processes now has access to all the memory of an entire node. (e) As in (c), but here each process uses OpenMP to run two software threads.

When a process is assigned a number of cores, it is possible to use OpenMP to split the computational work among several execution threads that share memory. In Fig. A1e, each of four processes has access to two software threads that share memory. In Figs. A1b and A1e, the number of software threads (i.e., eight) is equal to the number of available cores. These applications could well be equally efficient.

On a large cluster, with many nodes, it is necessary to use MPI if a single program is to use a large number of cores. Using only OpenMP, a program cannot use more than one node. When it is not possible to split the numerical problem, using MPI, into a large number of quasi-independent subprograms with modest memory requirements, it may be more efficient to use both MPI and OpenMP. For intermediate programs, which can run on one node, one can use either MPI or OpenMP. The Canadian EnKF was first parallelized using MPI (Houtekamer and Mitchell 2001). At an early stage, the additional use of OpenMP had been implemented but was not retained, since using only MPI was equally efficient. For now, we can run the operational configuration, as in Fig. A1b, with each process using only one core. In the context of the weak scaling experiments in this paper, however, we had to reintroduce OpenMP directives to run the larger configurations. Obviously, it takes a certain amount of development work to move an application from a single type of parallelization to two types of parallelization. Normally, after some additional coding effort, the hybrid code will become more efficient than the pure MPI code. In the future, we expect to have access to a larger number of cores and so an efficient use of OpenMP, in addition to MPI, may become critically important.

APPENDIX B

Computer Programs

To run a data assimilation cycle with the EnKF, we use a set of 15 individual programs that share software libraries. The 12 most important of these (i.e., with execution times of at least 1 s) perform the following distinct tasks:

B1—compute the mean and standard deviation of an ensemble of meteorological fields;
B2—generate an ensemble of random isotropic perturbation fields with correlations described by the background-error covariance matrix used in CMC's global variational system; these perturbation fields are the dominant component of the EnKF's model-error description (Houtekamer et al. 2009);
B3—interpolate perturbation fields to a different horizontal grid;
B4—interpolate perturbation fields to different vertical levels and add to a preexisting ensemble of fields;
B5—convert input observation files into a corresponding object;
B6—apply the forward operator to an ensemble of fields (details given in section 4a);
B7—assign a height in the atmosphere to each observation (for the vertical covariance localization);
B8—compare the observation and the prior as part of the quality control procedure for satellite winds;
B9—order the observations for sequential processing (details given in section 4a);
B10—compute the analysis increments and add to the background ensemble (this program is discussed extensively in section 4b);
B11—convert the final observation object into input files for an SQLite observation database; and
B12—obtain a subensemble of 20 analyses to be used as initial conditions for medium-range forecasts.

Fourteen of these 15 programs (the single exception being program B2) are included in a single code repository that also contains a number of FORTRAN modules. At the time of this writing, the repository contains about 26 000 lines of FORTRAN code. (In fact, almost 10 000 of these lines are either comments or are blank.) This line count includes neither the RTTOV radiative transfer code package (Saunders et al. 1999), version 8.7 of which we use in the forward operator for the radiance observations, nor the forward operator for the global positioning system-radio occultation (GPS-RO) observations (Aparicio et al. 2009).

In our experience, having a number of distinct programs permits the simultaneous research into and development of different aspects of the EnKF. A major current objective is to arrive at a set of shared modules that can be used both in the EnKF and the ensemble–variational analysis systems. To arrive at this end, we are gradually refactoring these two functioning data assimilation systems (Fowler 1999).

APPENDIX C

Communication of $\mathcal{H}(\Psi)$

For the various scaling experiments, we obtained execution times for the communication of $\mathcal{H}(\Psi)$. From Tables 3, 6, and 7, we generally note that execution times increase with the number of processes N_p.

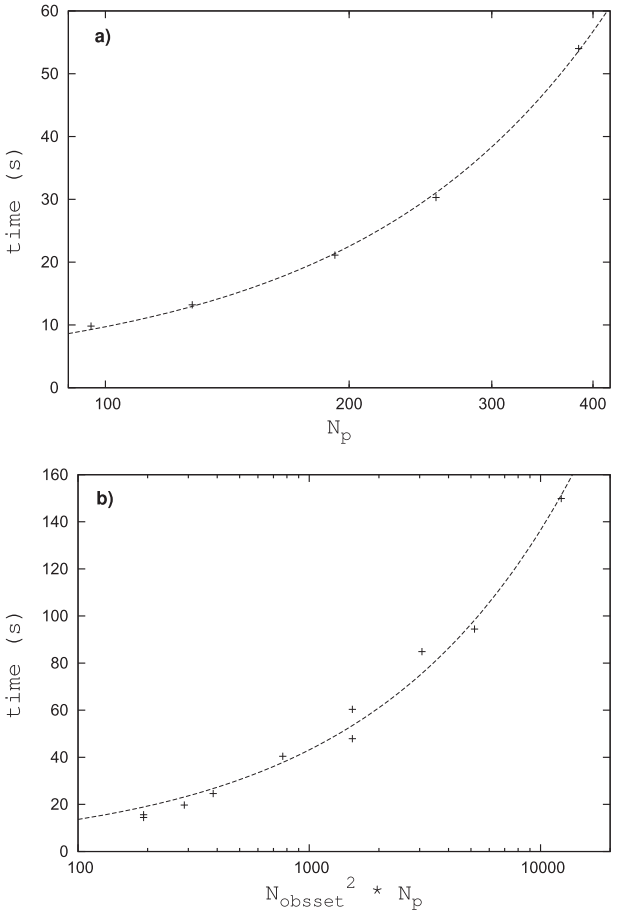

FIG. C1. The time needed to communicate $\mathcal{H}(\Psi)$. (a) Results for the weak scaling with ensemble size are shown. (b) Results for the strong scaling experiments, which have $N_{\text{obsset}} = 1$, are shown together with those for the weak scaling with the number of observations for which we have $N_{\text{obsset}} = 1$, 2, 3, and 4. The fitting functions are discussed in this appendix.

To fit, in Fig. C1a, the results of Table 6, we use the empirical equation

$$t = (aN_{\text{ens}} + b)N_p^{0.5}, \tag{C1}$$

where a and b are tunable parameters.

A reasonable fit, in Fig. C1b, to the results of Tables 3 and 7, could be obtained with the one-parameter empirical equation:

$$t = aN_{\text{obsset}}N_p^{0.5}, \tag{C2}$$

where a is a tunable parameter and N_{obsset} equals the number of observation sets. As a first approximation, it is reasonable to assume that the communication time is proportional to N_{obsset}. We did not manage to fit all numerical results with a single empirical equation.

Equations (C1) and (C2) suggest that the communication time t is proportional to $N_p^{0.5}$. This apparent square root dependence is related to the relatively small number of observations that are assimilated per observation batch and to the initial round-robin distribution of values of $\mathcal{H}(\Psi)$ over the processes. As N_p increases, the likelihood of a process having no values of $\mathcal{H}(\Psi)$ to communicate increases as well. This reduced need for communication leads to a reduced cost. With substantially larger batch sizes, we would a priori expect a less favorable scaling proportional to N_p because all processes would have information to communicate.

Conversely, a possible optimization of the algorithm would be, at the initial distribution of the extended part of the state vector, to send all the values of $\mathcal{H}(\Psi)$ that are to be used in a specific batch to the same process. Then, when assimilating a batch, only one process would need to send information. Since the number of batches is typically of the same order as N_p and since the extended part of the state vector (of length N_{obs}) is much smaller than the model state itself (length N_{model}), this should not overly affect the now nearly perfect scaling obtained for the analysis update computations (Table 3). Indeed, preliminary strong scaling results show that with this revised distribution strategy, the cost of communicating $\mathcal{H}(\Psi)$ is reduced to approximately 7 s independent of N_p.

REFERENCES

Amdahl, G. M., 1967: Validity of the single processor approach to achieving large-scale computing capabilities. *Spring Joint Computing Conf.,* Atlantic City, NJ, American Federation of Information Processing Societies, 483–485.

Anderson, J. L., 2001: An ensemble adjustment Kalman filter for data assimilation. *Mon. Wea. Rev.,* **129,** 2884–2903.

——, and N. Collins, 2007: Scalable implementations of ensemble filter algorithms for data assimilation. *J. Atmos. Oceanic Technol.,* **24,** 1452–1463.

Aparicio, J. M., G. Deblonde, L. Garand, and S. Laroche, 2009: Signature of the atmospheric compressibility factor in COSMIC, CHAMP, and GRACE radio occultation data. *J. Geophys. Res.,* **114,** D16114, doi:10.1029/2008JD011156.

Benkner, S., D. F. Kvasnicka, and M. Lucka, 2002: Experiments with Cholesky factorization on clusters of SMPs. *Proc. European Conf. on Numerical Methods and Computational Mechanics,* University of Miskolc, Miskolc, Hungary, European Community on Computational Methods in Applied Sciences. [Available online at http://eprints.cs.univie.ac.at/1172/.]

Buehner, M., P. L. Houtekamer, C. Charette, H. L. Mitchell, and B. He, 2010a: Intercomparison of variational data assimilation and the ensemble Kalman filter for global deterministic NWP. Part I: Description of single-observation experiments. *Mon. Wea. Rev.,* **138,** 1550–1566.

——, ——, ——, ——, and ——, 2010b: Intercomparison of variational data assimilation and the ensemble Kalman filter for global deterministic NWP. Part II: One-month experiments with real observations. *Mon. Wea. Rev.,* **138,** 1567–1586.

Chandra, R., L. Dagum, D. Kohr, D. Maydan, J. McDonald, and R. Menon, 2001: *Parallel programming in OpenMP.* Academic Press, 230 pp.

Cohn, S. E., A. da Silva, J. Guo, M. Sienkiewicz, and D. Lamich, 1998: Assessing the effects of data selection with the DAO physical-space statistical analysis system. *Mon. Wea. Rev.,* **126,** 2913–2926.

Côté, J., S. Gravel, A. Méthot, A. Patoine, M. Roch, and A. Staniforth, 1998: The operational CMC–MRB Global Environmental Multiscale (GEM) model. Part I: Design considerations and formulation. *Mon. Wea. Rev.,* **126,** 1373–1395.

Courtier, P., J.-N. Thépaut, and A. Hollingsworth, 1994: A strategy for operational implementation of 4D-Var, using an incremental approach. *Quart. J. Roy. Meteor. Soc.,* **120,** 1367–1387.

Daley, R., 1991: *Atmospheric Data Analysis.* Cambridge University Press, 457 pp.

Evensen, G., and P. J. van Leeuwen, 1996: Assimilation of Geosat altimeter data for the Agulhas current using the ensemble Kalman filter with a quasigeostrophic model. *Mon. Wea. Rev.,* **124,** 85–96.

Fowler, M., 1999: *Refactoring: Improving the Design of Existing Code*. Addison-Wesley, 431 pp.

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

Girard, C., and Coauthors, 2014: Staggered vertical discretization of the Canadian Environmental Multiscale (GEM) model using a coordinate of the log-hydrostatic-pressure type. *Mon. Wea. Rev.,* **142,** 1183–1196.

Golub, G. H., and C. F. Van Loan, 1996: *Matrix Computations.* 3rd ed. The Johns Hopkins University Press, 694 pp.

Gropp, W., E. Lusk, and A. Skjellum, 1994: *Using MPI: Portable Parallel Programming with the Message-Passing Interface.* The MIT Press, 307 pp.

Hager, G., and G. Wellein, 2011: *Introduction to High Performance Computing for Scientists and Engineers.* Chapman and Hall/CRC, 330 pp.

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.

——, ——, and X. Deng, 2009: Model error representation in an operational ensemble Kalman filter. *Mon. Wea. Rev.,* **137,** 2126–2143.

——, X. Deng, H. L. Mitchell, S.-J. Baek, and N. Gagnon, 2014: Higher resolution in an operational ensemble Kalman filter. *Mon. Wea. Rev.,* **142,** 1143–1162.

Isaksen, L., 2012: Data assimilation on future computer architectures. *Proc. ECMWF Seminar on Data Assimilation for Atmosphere and Ocean,* Reading, United Kingdom, ECMWF, 301–322.

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.

Lorenc, A. C., 2003: The potential of the ensemble Kalman filter for NWP—A comparison with 4D-Var. *Quart. J. Roy. Meteor. Soc.,* **129,** 3183–3203.

Mandel, J., 2006: Efficient implementation of the ensemble Kalman filter. Center for Computational Mathematics Rep. 231, University of Colorado at Denver and Health Sciences Center, 9 pp. [Available online at math.ucdenver.edu/ccm/reports/rep231.pdf.]

Meuer, H., E. Strohmaier, J. Dongarra, and H. Simon, cited 2012: The Top500 list: Twenty years of insight into HPC performance. [Available online at www.top500.org/lists/2012/11.]

Mitchell, H. L., and P. L. Houtekamer, 2009: Ensemble Kalman filter configurations and their performance with the logistic map. *Mon. Wea. Rev.,* **137,** 4325–4343.

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

Nerger, L., and W. Hiller, 2013: Software for ensemble-based data assimilation systems—Implementation strategies and scalability. *Comput. Geosci.,* **55,** 110–118.

Saunders, R., M. Matricardi, and P. Brunel, 1999: An improved fast radiative transfer model for assimilation of satellite radiance observations. *Quart. J. Roy. Meteor. Soc.,* **125,** 1407–1425.

Szunyogh, I., E. J. Kostelich, G. Gyarmati, E. Kalnay, B. R. Hunt, E. Ott, E. Satterfield, and J. A. Yorke, 2008: A local ensemble transform Kalman filter data assimilation system for the NCEP global model. *Tellus,* **60A,** 113–130.

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.

——, ——, X. Wei, Y. Song, and Z. Toth, 2008: Ensemble data assimilation with the NCEP Global Forecast System. *Mon. Wea. Rev.,* **136,** 463–482.

Yang, S.-C., E. Kalnay, B. Hunt, and N. E. Bowler, 2009: Weight interpolation for efficient data assimilation with the local ensemble transform Kalman filter. *Quart. J. Roy. Meteor. Soc.,* **135,** 251–262.

Zhang, F., Y. Weng, J. A. Sippel, Z. Meng, and C. H. Bishop, 2009: Cloud-resolving hurricane initialization and prediction through assimilation of Doppler radar observations with an ensemble Kalman filter. *Mon. Wea. Rev.,* **137,** 2105–2125.