A Parallel Algorithm for Variational Assimilation in Oceanography and Meteorology

ANDREW F. BENNETT

College of Oceanography, Oregon State University, Corvallis, Oregon

JEROME R. BAUGH

Intel Supercomputer Systems Division, Beaverton, Oregon

(Manuscript received 24 June 1991, in final form 5 November 1991)

ABSTRACT

A parallel algorithm is described for variational assimilation of observations into oceanic and atmospheric models. The algorithm may be coded first for execution on a serial computer and then trivially modified for execution on a parallel computer such as the Intel iPSC/860. The speedup factor for parallel execution is roughly $P(2M+3)(2M+3P)^{-1}$, where P is the number of processors and M is the number of observations ($M \geq P$). The speedup factor approaches P from below as $M/P \to \infty$.

The algorithm has been applied in serial form to ocean tides (Bennett and McIntosh 1982; McIntosh and Bennett 1984; Bennett 1985) and oceanic equatorial interannual variability (Bennett 1990). It has been applied in parallel form to oceanic synoptic-scale circulation (Bennett and Thorburn 1992); a parallel application to operational forecasting of tropical cyclones is in progress (Bennett et al. 1992).

For the sake of simplicity, the parallel algorithm is described here for a model consisting of a linear, first-order wave equation with initial and boundary conditions plus a dataset consisting of observations at isolated points in space and time. However, measurements of parallel performance are given for a nonlinear quasigeostrophic model.

1. Introduction

Numerical models of oceanic and atmospheric circulation are now highly realistically detailed. Moreover, the forcing fields, initial fields, and boundary fields in the model are reasonably realistic estimates derived from significant quantities of data. Therefore, it is reasonable to compare the model solutions with data. However, the level of disagreement usually exceeds prior estimates of the level of measurement error. It would have to be concluded that either these prior estimates are wrong, or else the prior estimates of the forcing, of the initial values, or of the boundary values are wrong. The model dynamics could also be wrong, but in this formulation there is no distinction between dynamical error and forcing error.

Pseudorandom perturbations consistent with prior estimates of error statistics could be added to the various model inputs until an acceptable level of agreement with observations is reached. There are several possibilities: either no such perturbation exists, or at least one exists but none is found after a few trials, or else many are found. In each case the rational procedure would be to find the best fit to the model dynamics, the initial values, the boundary values, and the observations. The method of weighted least squares is the simplest fitting criterion. This strategy appears to have been discussed first by Sasaki (1955, 1970). If the observations were collected *before* the initial instant of the model, then the objective is one of *initialization,* that is, preparing the best estimate of initial values for the model consistent with the model dynamics and the previous observations. If the observations were collected *after* the model's initial instant, then the objective is *hindcasting,* that is, smoothing the observations in a manner consistent with the model dynamics. In either situation, it may be appropriate to require that the model dynamics (which include the prior forcing estimate) be satisfied exactly or only approximately; in the language of Sasaki, the model dynamics are to be imposed as either strong or weak constraints. A strong constraint may be achieved by assigning the model residual (see section 3) an arbitrarily large weight in the cost or penalty functional that defines the fitting criterion. Such a procedure is equivalent to appending the model dynamics to the rest of the functional using Lagrange multipliers.

There are several ways of finding the best fit, which is defined to be the minimum of the penalty functional. An elegant implementation of the method of gradient descent (Talagrand and Courtier 1987) is attracting considerable attention. However, it has not been applied to time-dependent, weak-constraint assimilation

Corresponding author address: Dr. Andrew F. Bennett, Oceanography Administration Building 104, Oregon State University, Corvallis, OR 97331.

problems, owing to poor conditioning of the descent algorithm when gradient vectors of very large dimension are involved. Preconditioning techniques exist but have yet to be successfully applied to such large problems. Calculating the error statistics for the best fit, that is, determining the posterior error covariances, has proven to be an impractical task using descent methods when the dimension of the gradient vector is very large. It should be possible to accelerate descent methods using parallel computers, but this does not appear to have been attempted yet.

Alternatively, the best fit may be found by solving the Euler–Lagrange (EL) equations that govern extrema of the penalty functional. The system of EL equations, initial conditions, and boundary conditions form a two-point boundary value problem in the time interval of interest. If the model is linear, then the associated EL problem is also linear. The solution of a linear EL problem can be expressed as a linear combination of the prior estimate of the solution (the response of the model to the prior estimates of its inputs) and the so-called representer functions associated with the assimilation problem. One representer function exists for each observation; the representers may be calculated without reference to each other, and these are the tasks that may be shared by multiple processors. The code modifications necessary for this sharing are so trivial that the efficiency of the parallel algorithm should be compared with serial implementations of alternatives: for example, single-processor codes for gradient descent. The representer functions also yield explicit expressions for the reductions in all of the error covariances as a consequence of data assimilation; that is, they yield the "explained" error covariances.

If the model is nonlinear, then so is the associated EL problem. Iterative methods may be used to reduce the nonlinear EL problem to a sequence of linear EL problems, each of which may be solved using linear combinations of corresponding prior solution estimates and representers. Much simpler iterative schemes may be devised, but they do not converge.

Descending the gradient and solving the EL equations are both variational methods for finding extrema of penalty functionals. Neither may be used if the model or observing system involves nonsmooth processes, such as discontinuous convective adjustment, vanishing layer thickness, or event recording.

The parallel algorithm has been applied iteratively to a nonlinear, reduced-gravity quasigeostrophic model in a doubly periodic domain, plus M pointwise observations of the streamfunction. The algorithm was coded in Fortran 77 and tested on a single-processor workstation. The addition of four nonstandard subroutine calls enabled the code, after recompilation, to execute on the Intel iPSC/860 computer at the Numerical Aerodynamic Simulation Facility (NAS), National Aeronautics and Space Administration (NASA) Ames Research Center. The computer has multiple-instruction multiple-data (MIMD) architecture, but was used in this application as though it were a single-instruction multiple-data (SIMD) machine. Additional tests were made on Intel iPSC/860 computers at the headquarters of Intel Corporation in Beaverton, Oregon, and at the California Institute of Technology in Pasadena, California. It will appear that the presentation concentrates almost entirely on the mathematics of the assimilation scheme, with only a brief description of the seemingly trivial parallel algorithm. However, it is the mathematical decomposition of the assimilation scheme into a number of logically identical tasks that is the key to the parallelization scheme.

The following description of the algorithm is based on a trivial model and observing system. Certain mathematical details, including the equations for the representers, are in the Appendix. Parallel performance figures are given for the nonlinear quasigeostrophic model.

2. The model and the observations

The model consists of the linear, first-order wave equation in one space dimension:

$$u_t + u_x = F + f, \tag{2.1}$$

where the "flow" $u = u(x, t)$ is the prototype state variable for oceanic or atmospheric circulation, $F = F(x, t)$ is a prior estimate of the model forcing, and $f = f(x, t)$ is the unknown error in that estimate. Subscripts denote partial derivatives with respect to time t and position x. A suitable initial condition at time $t = 0$ is

$$u(x, 0) = I(x) + i(x), \tag{2.2}$$

where I and i are prior initial estimate and error, respectively. The model domain is the interval $0 \leqslant x \leqslant L$; a suitable boundary condition at $x = 0$ is

$$u(0, t) = B(t) + b(t), \tag{2.3}$$

where B and b are the prior boundary estimate and error, respectively. No boundary condition is needed at $x = L$. Observations are of the form

$$d_m = u_m + \epsilon_m, \tag{2.4}$$

where $u_m = u(x_m, t_m)$ is a perfect measurement of the "true" flow at (x_m, t_m), ϵ_m is a measurement error, and d_m is a datum or recorded value. For definiteness, assume $0 \leqslant x_m \leqslant L$, and $0 \leqslant t_m \leqslant T$. That is, consider a hindcasting or *smoothing* problem. See Fig. 1 for a sketch of the domain and the data sites.

The prior estimate or first guess of the flow is $u_F = u_F(x, t)$, satisfying

$$(u_F)_t + (u_F)_x = F, \tag{2.5}$$

$$u_F(x, 0) = I(x), \tag{2.6}$$

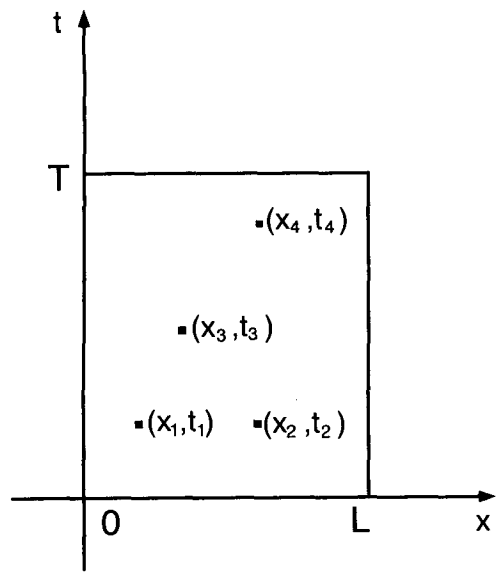

FIG. 1. Domain for the simple wave equation (2.1): $0 \leq x \leq L$, $0 \leq t \leq T$. Data are available at $M = 4$ points in the figure; in the actual computations M is as large as 128. The initial condition (2.2) holds at $t = 0$; the boundary condition (2.3) holds at $x = 0$.

and

$$u_F(0, t) = B(t). \tag{2.7}$$

The prior estimate of the data misfit is

$$h_m = d_m - u_F(x_m, t_m). \tag{2.8}$$

3. The penalty functional and the representer expansion

A simple penalty functional is provided by

$$J(u) = W_f \int_0^T \int_0^L (u_t + u_x - F)^2 dx dt + W_i \int_0^L (u - I)^2 dx + W_b \int_0^T (u - B)^2 dt + w \sum_{m=1}^{M} (d_m - u_m)^2, \tag{3.1}$$

where, for example, $(u_t + u_x - F)$ is the prior dynamical error or residual, while W_f, W_i, W_b, and w are positive weights. The rational choices for these weights are inverses of variances of the prior errors f, i, b, and ϵ_m, respectively.

The calculus of variations (Courant and Hilbert 1953) shows that $J(u)$ is least for $u = \hat{u}(x, t)$, satisfying

$$\hat{u}_t + \hat{u}_x = F + W_f^{-1}\lambda, \tag{3.2}$$

subject to the initial condition

$$\hat{u}(x, 0) = I(x) + W_i^{-1}\lambda(x, 0) \tag{3.3}$$

and to the left-hand boundary condition

$$\hat{u}(0, t) = B(t) + W_b^{-1}\lambda(0, t), \tag{3.4}$$

where the so-called adjoint variable λ satisfies the adjoint or Euler–Lagrange equation

$$-\lambda_t - \lambda_x = w \sum_{m=1}^{M} \delta(x - x_m)\delta(t - t_m)(d_m - \hat{u}_m), \tag{3.5}$$

subject to the final condition

$$\lambda(x, T) = 0 \tag{3.6}$$

and to the right-hand boundary condition

$$\lambda(L, t) = 0. \tag{3.7}$$

In (3.5), $\hat{u}_m = \hat{u}(x_m, t_m)$. It may be seen that the optimal estimates of the prior errors are

$$\hat{f}(x, t) = W_f^{-1}\lambda(x, t), \quad \hat{i}(x) = W_i^{-1}\lambda(x, 0), \quad \hat{b}(t) = W_b^{-1}\lambda(0, t). \tag{3.8}$$

In particular, it should be noted that the adjoint variable λ is not a Lagrange multiplier here; rather, it is *by definition* the weighted model residual: see (3.2). As a consequence of the presence of the term $\hat{u}_m$ on the right-hand side of (3.5), the system (3.3)–(3.7) is a coupled two-point boundary value problem in the interval $0 \leq t \leq T$, and *cannot* be solved by a backward integration of (3.5)–(3.7) followed by a forward integration of (3.2)–(3.4). However, the following strategy does solve (3.2)–(3.7) with a finite number of backward and forward integrations. It may be shown that $\hat{u}$ can be expressed as

$$\hat{u}(x, t) = u_F(x, t) + \sum_{m=1}^{M} \hat{\beta}_m r_m(x, t). \tag{3.9}$$

In (3.9), u_F is the prior flow estimate, while the M functions r_m are the representer functions for (3.2)–(3.7) (see the Appendix). Substitution of (3.9) into (3.2)–(3.7), followed by a comparison with (A.1)–(A.6) for the representers and their adjoints, shows immediately that the M coefficients $\hat{\beta}_m$ satisfy the finite-dimensional linear system

$$\sum_{m=1}^{M} (R_{nm} + w^{-1}\delta_{nm})\hat{\beta}_m = h_n, \tag{3.10}$$

where the h_n are, again, the prior data misfits, while w is the data weight and R_{nm} is the nth measurement of the mth representer:

$$R_{nm} = r_m(x_n, t_n), \quad 1 \leq n, \quad m \leq M. \tag{3.11}$$

It may also be shown that $R_{nm} = R_{mn}$ and, provided $(x_n, t_n) \neq (x_m, t_m)$ for any $n \neq m$ (that is, no repeated observations), the matrix $\mathbf{R}$ is positive definite.

The mth representer r_m is calculated in two stages (see Appendix). The first is the integration of an inhomogeneous wave equation backward in time to $t = 0$ from a homogeneous "final condition" at $t = T$, subject to a homogeneous boundary condition at $x = L$

[see (A.4)–(A.6)]. The backward wave problem is forced by an impulse at (x_m, t_m). The second step is the integration of an inhomogeneous wave equation forward in time from an inhomogeneous initial condition at time $t = 0$, subject to an inhomogeneous boundary condition at $x = 0$ [see (A.1)–(A.4)]. The inhomogeneities in the second integration are all values of the solution of the backward or "adjoint" problem, divided by the corresponding weights.

It would appear from (3.9) that the evaluation of the posterior estimate $\hat{u}$ requires the storage of the M representers as each is calculated, and then forms the sum as shown. However, both these steps may be avoided by using the result

$$\hat{\epsilon}_m = d_m - \hat{u}_m = w^{-1}\hat{\beta}_m, \tag{3.12}$$

where $\hat{\beta}_m$ is the mth representer coefficient. Substituting (3.12) into the adjoint equation (3.5) decouples that equation from the forward problem (3.2)–(3.4), so (3.5)–(3.7) may be integrated backward in time yielding $\lambda(x, t)$. A subsequent integration of (3.2)–(3.4) forward in time yields $\hat{u}(x, t)$ for all x and t. Thus, the total number of model integrations is $2M + 3$: 1 for the prior estimate u_F [see (2.5)–(2.7)], 2 for each of M representers, 1 for λ, and 1 for $\hat{u}$.

4. The posterior error covariances

Assuming that the prior errors f, b, i, and ϵ_m are random, have vanishing means, are uncorrelated, and have autocovariances that are the inverses (see the Appendix) of the respective weights in the quadratic penalty function (3.1), it may be shown that the error covariance for the posterior flow estimate is

$$\begin{aligned} C_{\hat{u}\hat{u}}(x, t, x', t') &\equiv \overline{[u(x, t) - \hat{u}(x, t)][u(x', t') - \hat{u}(x', t')]} \\ &= C_{uu}(x, t, x', t') - \sum_{n,m=1}^{M} r_n(x, t) K_{nm} r_m(x', t'), \end{aligned} \tag{4.1}$$

where C_{uu} is the error covariance for the prior flow estimate:

$$\begin{aligned} C_{uu}(x, t, x', t') &\equiv \overline{[u(x, t) - u_F(x, t)][u(x', t') - u_F(x', t')]} \end{aligned} \tag{4.2}$$

and $\mathbf{K}$ is the inverse of the matrix $(\mathbf{R} + w^{-1}\mathbf{I})$. Similarly, the error covariance for the posterior boundary estimate is

$$\begin{aligned} C_{\hat{b}\hat{b}}(t, t') &\equiv \overline{[b(t) - \hat{b}(t)][b(t) - \hat{b}(t')]} \\ &= C_{bb}(t, t') - \sum_{n,m=1}^{M} r_n(0, t) K_{nm} r_m(0, t'), \end{aligned} \tag{4.3}$$

where C_{bb}, the error covariance of the prior boundary estimate is given here by

$$C_{bb}(t, t') \equiv \overline{b(t)b(t')} = W_b^{-1}\delta(t - t'). \tag{4.4}$$

Analogous formulas hold for the errors covariances for the posterior initial estimate, the posterior forcing estimate, and the posterior data misfits. With the exception of C_{uu} in (4.1), every term in these formulas is either a specified prior parameter such as W_b^{-1} or else is a combination of representers.

The missing covariance C_{uu} is most easily calculated by statistical simulation, that is, by solving (2.1)–(2.3) repeatedly, with independent pseudorandom samples for the errors f, i, and b.

It is necessary to store the representers if the posterior covariances are required. An exception is the posterior measurement error covariance, which depends only on w and $\mathbf{K}$. However, the effort in calculating and storing the representers is rewarded with *explicit* formulas for the posterior error covariances. However, even the representers need not be stored if the linear algebra is exploited to the full. Diagonalizing the matrix $\mathbf{R}$ and projecting the representer coefficients $\hat{\beta}$ onto the eigenvectors of $\mathbf{R}$ (that is, defining new measurements as linear combinations of the original measurements) yields new representers for which the quadratic forms in (4.1) and (4.2) are diagonal. Then the sums in (4.1) and (4.2) can be calculated without having to store all the representers simultaneously. The price is that the new representers have to be computed by integrating analogs of (A.1)–(A.6), with linear combinations of impulses replacing the individual impulses on the right-hand side of (A.4). It is usually the case that a majority of the eigenvalues of $\mathbf{R}$ are smaller than w^{-1} [see (3.10)], so in practice most of the new representers need not be computed. In summary, the representers need never be stored. All of the old representers must be computed; a minority of the new representers must also be computed if posterior error covariances are required.

5. The parallel algorithm

Determining the optimal flow estimate $\hat{u}$ has been reduced to integrating a finite number of initial-value problems. In practice, a numerical integration technique is used, such as a finite-difference method based on a space–time grid.

The calculation could be accelerated using a parallel computer by decomposing the spatial domain into subdomains, each of which is allocated to a single processor. Integration would require sharing updated flow values at internal or subdomain boundaries. The "message passing" that results in the data sharing would lead to a great expansion of the code, with special cases for external or domain boundaries.

A completely different approach is taken here. In essence, each processor is used to calculate one representer in its entirety without domain decomposition.

Therefore, it is necessary that each processor have sufficient speed and local memory to facilitate such a calculation. The parallel algorithm has several versions; the easiest version to code, if somewhat redundant, is as follows.

Step 1. Each processor performs one forward integration for the prior estimate $u_F(x, t)$, keeping all the M prior data misfits $h_m = d_m - u_F(x_m, t_m)$, $1 \leqslant m \leqslant M$, in each local memory.

Step 2. Each processor calculates one representer. That is, the mth processor calculates $r_m(x, t)$, requiring one backward and one forward integration. The M measurements $r_m(x_n, t_n)$, $1 \leqslant n \leqslant M$, are kept in the mth local memory. These measurements compose the mth column of the representer matrix.

Step 3. Each processor sends its column to the other $M - 1$ processors.

Step 4. Each processor receives the other $M - 1$ columns.

Step 5. Each processor assembles the full representer matrix **R**.

Step 6. Each processor determines the representer coefficients $\hat{\beta}_1, \cdots, \hat{\beta}_M$, and hence the posterior data misfits $\hat{\epsilon}_m \equiv d_m - \hat{u}_m = w^{-1}\hat{\beta}_m$, $1 \leqslant m \leqslant M$.

Step 7. Each processor makes a backward integration to find the adjoint variable λ, and then a forward integration for the posterior flow estimate $\hat{u}$.

A number of remarks can be made about this algorithm.

Point 1. Each processor executes logically identical code; thus, the algorithm is ideal for a SIMD computer. The most widely available such computer is the Connection Machine (Hillis 1985), but the current model (the CM-2) uses relatively slow processors ($\leqslant$0.1 Mflops) with limited local memory ($\leqslant$1 128 kbyte). Moreover, it uses a proprietary version of Fortran 90 (Metcalf and Reid 1990) that is better suited to domain decomposition than to the aforementioned task decomposition. However, the parallel algorithm is easily implemented on current MIMD computers such as the Intel iPSC/860. These systems have up to 128 relatively fast processors. Each has a peak performance of 80 Mflops (single precision) and 60 Mflops (double precision); each can have up to 64 Mbytes of local memory. The algorithm may be coded in serial form in standard Fortran 77, and then converted to parallel form by the addition of four proprietary Fortran subroutine calls.

(i) The first call identifies the processor number in step 2, in the range 0–127.

(ii) The second sends one column to all other processors (step 3).

(iii) The third receives all other columns (step 4).

(iv) The fourth identifies the received columns (step 5).

If the number M of observations exceeds the number P of processors, then each processor must calculate at least M/P representers. This requires some additional "bookkeeping," which may be written in standard Fortran 77. The number P of available processors may either be read from a file or else determined by a proprietary Fortran call.

Point 2. If $M \leqslant P$, then the number of sequential integrations in the parallel algorithm is 5, consisting of 1 for u_F, 2 for the representers r_m, $1 \leqslant m \leqslant M$, and 2 for $\hat{u}$. Hence, the speedup factor is $(2M + 3)/5$, assuming that each integration takes equal time. Allowances will be made for time differences in the discussion of the results in section 6. If M/P is an integer greater than unity, then the sequential integration count is $(2M/P) + 3$, for a speedup factor of $P(2M + 3)(2M + 3P)^{-1}$. The factor approaches P from below as $M/P \to \infty$.

Point 3. The adjoint variable for the mth representer vanishes identically for $t_m \leqslant t \leqslant T$. Hence, the backward integration may be started at $t = t_m$ (see Appendix). This is an important saving in the serial algorithm, but saves no time in the parallel algorithm. The assembly of the representer matrix cannot be completed until all the representers have been calculated. The longest such calculation will, of course, be that corresponding to the largest value of t_m. On the other hand, staggering the completion of the representer calculations may alleviate bottlenecks if the representers are to be stored off-line for the subsequent calculation of posterior error covariances.

Point 4. The adjoint variables computed by backward integration may be overwritten by the respective "forward" variables r_m and $\hat{u}$ during forward integration.

Point 5. The messages passed between the processors are all vectors of length M ($4M$ bytes, in single precision). Since each processor sends a message to every other, there are in principle $M(M - 1)$ messages, including copies, or $4M^2(M - 1)$ bytes ($M \leqslant P$). The message volume is $4MP(P - 1)$ bytes at any one time, if $M > P$.

Point 6. While each processor must compute or read the prior estimates F, I, B, and d_m, $1 \leqslant m \leqslant M$, only one processor need compute and write the posterior estimates $\hat{u}$, $F + \hat{f}$, $I + \hat{i}$, $B + \hat{b}$, and $\hat{u}_m$, $1 \leqslant m \leqslant M$.

Point 7. No parallel algorithm is offered here for the evaluation of the posterior error covariances. However, the statistical estimation of the prior error covariance C_{uu} for the flow may be accelerated in the obvious way: each processor computes an independent sample of u. The speedup factor is naturally P.

Point 8. The speedup factor for calculating $\hat{u}$ is less than P, regardless of the value of M. Denoting the factor by S, one has $S \sim 2M/5$ when $1 \ll M \leqslant P$, for example. This can be substantial; $S \approx 50$ when $M = 128$. However, if there are N assimilations to be performed, say, in an experiment with an archive of N datasets, and $S < N$, then it is more efficient to have each processor execute the entire serial algorithm (that

is, each processor calculates every representer). This simple approach is available, whatever variational assimilation scheme is in use. For example, a large number of applications of a serial-coded descent algorithm could also, of course, be accelerated by using many processors. The corresponding speedup factor is the lesser of N and P, regardless of assimilation scheme.

Point 9. The parallel algorithm used here was chosen mainly for ease of comprehension and ease of implementation. It is not, however, the most efficient way to use a MIMD machine. For small P, a scheduling strategy would give better performance for a modest perturbation to the existing code.

The strategy addresses the inefficiency discussed in point 3, namely, that the larger the value of the measurement time t_m, the longer it takes to calculate the representer r_m. This leads to some processors being idle while others complete their tasks. Thus, the optimal strategy would be to schedule the representer calculations so that each processor calculates a mix of representers. The mixes would, of course, be chosen so that the total time taken is the same for each processor. In situations where the execution times for the distributed tasks cannot be reliably anticipated, a "manager–worker" strategy may be effective. One processor is reserved as a manager: it does no work. The other processors report to the manager for further work after completing an assigned task.

6. Results

The algorithm illustrated in the preceding sections has been applied iteratively to a nonlinear, reduced-gravity quasigeostrophic model in a doubly periodic, square region (Bennett and Thorburn 1992). The data consisted of streamfunction measurements at M points isolated in space and time, with $M \leqslant 128$. Iteration on the nonlinear Euler–Lagrange equations yielded a sequence of linear Euler–Lagrange equations. The dependent variables at the nth level of iteration were advected by the $(n-1)$th velocity estimate; a quadratic term arising from functional variation of the advecting velocity was evaluated entirely in terms of $(n-1)$th estimates. See Bennett and Thorburn (1992) for details and further discussion of the representer method.

Some timing results are shown in Fig. 2, the number of measurements being $M = 64$ and the number of processors P lying in the range $1 \leqslant P \leqslant 64$. The expected execution time *per iteration* using P processors is

$$T_P = F + (M/P)R + E, \qquad (6.1)$$

where F is the time to calculate the prior estimate (u_F in the prototype problem considered here), R is the time to calculate a representer (r_m here), and E is the time to calculate the posterior estimate ($\hat{u}$ here). Again, this last task involves integrating the original Euler–Lagrange equations [(3.2)–(3.7) here] after explicitly

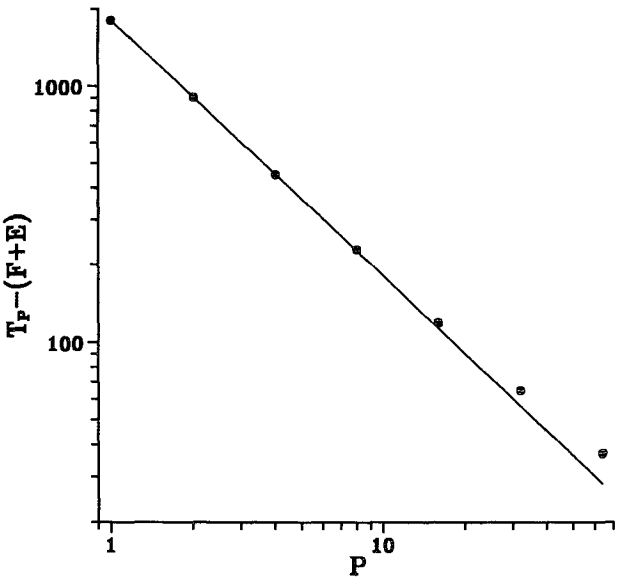

FIG. 2. Log–log plot of $T_P - (F + E)$ versus P, where P is the number of processors ($1 \leqslant P \leqslant 64$), $(F + E)$ is the time to execute the serial parts of the code, and T_P is the total execution time. The difference is the parallel execution time; it is expected to be inversely proportional to P. The slope of the straight line is -1.

evaluating the misfits [$(d_m - \hat{u}_m)$ in (3.5) here] using the representer coefficients ($\hat{\beta}_m$ here). Note that E includes the negligible time to solve for the representer coefficients [see (3.10) here]. The values for F and E were 39 and 46–49 s, respectively. Fig. 2 shows that $T_P - (F + E)$ is indeed inversely proportional to P. The average value of R is about 28 s. The execution times per iteration are also listed in Table 1. Note that although only $(F + E)/T_1 \approx 4\%$ of the original code remains in serial form, the speedup factor for $P = 64$ is $S = T_1/T_{64} \approx 15$. Amdahl's law prevails: if half the code can be run at infinite speed, then the whole code runs twice as fast (see, for example, Levesque and Williamson 1989). Recall, however, that $S \to P$ as $M/P \to \infty$. As Table 1 shows, the speedup factor S is close to 4 when $P = 4$ (and $M/P = 16$).

7. Summary

We have described a parallel algorithm for variational assimilation of observations into oceanic and atmospheric circulation models. The parallel form of the algorithm is very easily developed from the serial form; only four nonstandard Fortran statements are needed once the serial form has been coded in standard Fortran 77. The algorithm may be applied directly to linear models and may be applied iteratively to nonlinear models. Explicit formulas are available for the posterior error covariances. The theoretical speedup factor for a single assimilation is $S = P(2M + 3)(2M + 3P)^{-1}$, when there are M observations and P processors ($M \geqslant P$). Execution times for a quasigeo-

TABLE 1. Execution times (s) for the entire code (T_P), and for the parallel part of the code ($T_P - F - E$) versus the number of processors P.

P	T_P	$T_P - (F + E)$
1	1896	1811
2	989	904
4	536	451
8	314	229
16	204	119
32	150	65
64	125	37

strophic model indicate smaller speedup factors, owing to the fact that the representer integrations take less time than the other integrations. The parallel algorithm is most efficient ($S \approx P$) when M/P is large. This will be the case of greatest practical interest for existing large atmospheric datasets and as real-time remote sensing of ocean circulation from satellites becomes more important. The serial form of the algorithm, running independently on each processor of a parallel computer, may be more the efficient if the number of assimilations exceeds S.

We have emphasized the simplicity and convenience of being able to develop the algorithm in serial form, using conventional computing hardware and software, prior to an easy port to parallel computers of MIMD design. However, the algorithm may also be accelerated by the use of SIMD computers. The simplest approach would be to calculate each representer in turn. Speed would be gained by exploiting the array-processing capabilities of SIMD machines, provided the model physics and the numerical methods lend themselves to expression in terms of simple array operations. Quasigeostrophic models, balance equation models, and primitive equation models subject to the rigid-lid approximation all require the solution of elliptic problems in order to find the streamfunction. Solving these problems reduces, in practice, to inverting large matrices. Implicit integration schemes, for efficient integration of the external gravity mode in free-surface models, also require the inversion of large matrices. While matrix inversion can be reduced to simple array operations, it is not an ideal task for a SIMD machine. Moreover, these machines cannot be programmed in Fortran 77, and so workstation codes cannot be ported. Nevertheless, there is at least one ocean circulation model that is ideally suited to a SIMD machine. The model, owing to Bleck and Smith (1990), is of primitive equation type and has a free surface. Moreover, both the internal and external modes are advanced using explicit time stepping. Interprocessor communication is restricted to nearest neighbors only, and so efficient code is readily developed in Fortran 90. To summarize, certain ocean models and numerical methods are well suited to SIMD machines, so variational assimilation into these models using representer methods could be accelerated simply by recoding the serial algorithm in Fortran 90. In comparison, the approach that we have described renders both the choice of model physics and the choice of numerical method independent of the computer architecture. The approach also obviates the need for programming languages with array-processing capabilities.

It must be emphasized that if domain decomposition is to be avoided entirely, then the approach described here requires each processor in a MIMD computer to have sufficient local memory for a *single,* scalar space–time field, such as $u(x, t)$, $\lambda(x, t)$, or $r_m(x, t)$, and be fast enough to compute the field in a convenient amount of time. Of course, in a more sophisticated model it may be necessary to store a single *vector* field. As already mentioned, the processors in the Intel iPSC/860 can have up to 64 Mbytes of local memory and have a peak speed of 80 Mflops (single precision). These specifications are inadequate for long-term integration of time-dependent, eddy-resolving, basin-scale, or global primitive equation models. However, this is not a real problem at present, and is unlikely to be so for some time, owing to the lack of observations of sufficient density and coverage. While satellite altimeters may shortly provide adequate surveys of sea level elevation, comparable subsurface data are not anticipated in the near future. Hence, a data-assimilating simulation of the actual global ocean eddy field, as distinct from a simulation of a statistically similar field, is not yet feasible. Thus, on one hand, data-assimilating simulations of the actual ocean eddy field are perforce limited to regions and times of availability of eddy-resolving subsurface data, while on the other hand data assimilation can at present only influence the larger-scale fields in basin-scale or global models. In this latter context, it is essential to note that the representers are covariances: here they are the covariances of the flows $u(x, t)$ driven by random forcing fields $f(x, t)$, random initial values $i(x)$, and random boundary values $b(t)$, all having first and second moments as specified in the Appendix [see (A.7)–(A.12)]. That is, $r_m(x, t) = [u(x, t) - u_F(x, t)][u(x_m, t_m) - u_F(x_m, t_m)]$; see Bennett (1990) for a proof and further discussion. In general, the representers will be much smoother than are individual realizations of the flow. Therefore, it is not necessary to calculate the representers with the same resolution in space and time as used for the first guess $u_F(x, t)$ or posterior estimate $\hat{u}(x, t)$. Owing to efficient communications between computers, it is now feasible to calculate u_F and $\hat{u}$ on a large-memory, high-performance serial processor or on a massively parallel SIMD processor by domain decomposition, and to calculate the representers on individual processors of a MIMD machine with relatively small amounts of local memory. The compromise is in fact minimal, if steps 6 and 7 of the algorithm (see section 5) are followed. That is, the representers are used only to calculate the posterior data misfit $d_m - \hat{u}_m$ appearing in

the adjoint equation (3.5). Subsequent backward integration of (3.5)–(3.7) for λ, and forward integration of (3.2)–(3.4) for $\hat{u}$, has the effect of integrating the representers in (3.9) at high resolution; the only compromise is in the calculation of the representer coefficients $\hat{\beta}_m$.

We conclude by remarking that the widespread adoption of variational assimilation schemes, especially in the operational context, will depend upon the development of algorithms that exploit sophisticated computer architectures. The parallel algorithm described here is offered as a promising example.

Acknowledgments. We thank the Numerical Aerodynamic Simulation (NAS) Facility at NASA/Ames and the Caltech Concurrent Supercomputer Facility at the California Institute of Technology for the use of their Intel iPSC/860 computers. The help of Leigh-Ann Tanner at NAS and Heidi Lorenz-Wirzba at CCSF is especially appreciated. One of us (AFB) is supported by the Office of Naval Research (Grant N00014-90J1142). Lance Leslie suggested the simple wave equation for an illustration of variational assimilation. Carl Hagelberg ran the codes on the machines at NAS and Caltech. Barbara McVicar typed the manuscript, which was considerably improved following a careful reading by Jack Barth.

APPENDIX

Representers and Prior Covariances

a. Representers

The mth representer $r_m = r_m(x, t)$ satisfies

$$(r_m)_t + (r_m)_x = W_f^{-1}\alpha_m, \tag{A.1}$$

subject to the initial condition

$$r_m(x, 0) = W_i^{-1}\alpha_m(x, 0) \tag{A.2}$$

and to the left-hand boundary condition

$$r_m(0, t) = W_b^{-1}\alpha_m(0, t). \tag{A.3}$$

The adjoint representer variable $\alpha_m = \alpha_m(x, t)$ satisfies

$$-(\alpha_m)_t - (\alpha_m)_x = \delta(x - x_m)\delta(t - t_m), \tag{A.4}$$

subject to the final condition

$$\alpha_m(x, T) = 0 \tag{A.5}$$

and to the right-hand boundary condition

$$\alpha_m(L, t) = 0. \tag{A.6}$$

b. Prior covariances

It is assumed that the prior errors f, b, i, and ϵ_m in the forcing, the boundary values, the initial values, and the data, respectively, have vanishing means:

$$\bar{f} = \bar{b} = \bar{i} = \bar{\epsilon}_m = 0, \quad 1 \leqslant m \leqslant M; \tag{A.7}$$

the prior errors are mutually uncorrelated:

$$\overline{fb} = \overline{fi} = \overline{f\epsilon}_m = \overline{bi} = \overline{b\epsilon}_m = \overline{i\epsilon}_m = 0; \tag{A.8}$$

and have the following autocovariances:

$$\overline{f(x, t)f(x', t')} = W_f^{-1}\delta(x - x')\delta(t - t'), \tag{A.9}$$

$$\overline{b(t)b(t')} = W_b^{-1}\delta(t - t'), \tag{A.10}$$

$$\overline{i(x)i(x')} = W_i^{-1}(x - x'), \tag{A.11}$$

and

$$\overline{\epsilon_m\epsilon_n} = w^{-1}\delta_{n,m}. \tag{A.12}$$

More general forms for the covariances would require a more general form for the penalty functional (3.1).

REFERENCES

Bennett, A. F., 1985: Array design by inverse methods. *Progr. Oceanogr.*, **15**, 129–156.

——, 1990: Inverse methods for assessing SOP networks and estimating circulation and winds from tropical XBT data. *J. Geophys. Res.*, **95**, 16 111–16 148.

——, and P. C. McIntosh, 1982: Open ocean modeling as in inverse problem: Tidal theory. *J. Phys. Oceanogr.*, **12**, 1004–1018.

——, and M. A. Thorburn, 1992: The generalized inverse of a nonlinear quasigeostrophic ocean circulation model. *J. Phys. Oceanogr.*, **22**, 213–230.

——, L. M. Leslie, C. R. Hagelberg, and P. E. Power, 1992: Tropical cyclone data assimilation experiments using a barotropic model initialized by a generalized inverse method. *Mon. Wea. Rev.*, (submitted).

Bleck, R., and L. T. Smith, 1990: A wind-driven isopycnic coordinate model of the North and equatorial Atlantic Ocean: 1. Model development and supporting experiments. *J. Geophys. Res.*, **95**, 3273–3286.

Courant, H., and D. Hilbert, 1953: *Methods of Mathematical Physics.* Vol. 1. Wiley Interscience, 561 pp.

Hillis, D. W., 1985: *The Connection Machine.* The MIT Press, 190 pp.

Levesque, J. M., and J. W. Williamson, 1989: *A Guide to Fortran on Supercomputers.* Academic Press, 218 pp.

McIntosh, P. C., and A. F. Bennett, 1984: Open ocean modeling as an inverse problem: Bass Strait tides. *J. Phys. Oceanogr.*, **14**, 601–614.

Metcalf, M., and J. Reid, 1990: *Fortran 90 Explained.* Clarendon, 294 pp.

Sasaki, Y., 1955: A variational study of the numerical prediction based on the variational principle. *J. Meteor. Soc. Japan*, **33**, 262–275.

——, 1970: Some basic formalism in numerical variational analyses. *Mon. Wea. Rev.*, **98**, 43–60.

Talagrand, O., and P. Courtier, 1987: Variational assimilation of meteorological observations with the adjoint vorticity equation. I: Theory. *Quart. J. Roy. Meteor. Soc.*, **113**, 1311–1328.