# Atmospheric Data Assimilation on Distributed-Memory Parallel Supercomputers

Chris H. Q. Ding[1], Peter M. Lyster[2], Jay W. Larson[2], Jing Guo[3], Arlindo da Silva[3]

[1] National Energy Research Scientific Computing Center
Lawrence Berkeley National Laboratory
University of California, Berkeley, CA 94720, USA

[2] Interdisciplinery Center for Earth Systems Science
University of Maryland, College Park, MD 20742, USA

[3] Data Assimilation Office, Goddard Space Flight Center
Greenbelt, MD 20071, USA

**ABSTRACT**. Atmospheric data such as temperature, moisture, winds, etc., collected by satellites and direct measuements from upper-air instruments, ground observation stations provide only partial information about the atmosphere. They are assimilated to numerical forecasts to provide a coherent, evolving state of the global atmosphere. The data analysis system, the Physical-space Statistical Analysis System (PSAS) developed at the Data Assimilation Office at NASA's Goddard Space Flight Center, requires computing resources far beyond the capabilities of even the state-of-the-art vector supercomputers. Here we describe an efficient and scalable implementation of the PSAS on distributed-memory massively parallel supercomputers such as Intel Paragon and Cray T3E; the implementations achieves superb performance as demonstrated by detailed performance analysis of systematic runs on up to 512 processors on Paragon, T3D and T3E. Consequently, the solution time is reduced to 24.6 seconds on 512-PE T3E from 5 hours on a single head of Cray C90 for a real problem of 80,000 observations, a 740-fold reduction of turn-around time.

## 1. Introduction

Atmospheric data, such as temperature, winds, moisture, etc, have been routinely measured all over the Earth for several decades from diverse sources such as satellites, ground observation stations, or even flying balloons. These data sets are scattered over space and time in somewhat random ways, have incomplete global coverages (e.g., more data in North America), and suffer from many uncertainties, i.e., errors. Incorporating these observational data into the modeling systems is an important aspect of numerical weather prediction and climate modeling.

Numerical predictive models, such as the atmospheric general circulation models, are used to compute successive forecasts from a given initial condition. The forecast data sets are physically-consistent, complete, and time-contiguous. They are defined on regularly spaced gridpoints, usually $2^{o}$ in latitudes, $2.5^{o}$ on longitudes, at 20 or more vertical levels. However, errors of the model forecasts due to imperfect modeling mechanisms are hard to assess. Incorporating observation data into the numerical modeling process helps to correct these model errors and thus lead to more accurate predictions.

Figure 1 gives an outline of the data assimilation operations. Observation data of all types, winds, moisture etc., collected within a short time period are organized as a vec-

tor which represents the state of the weather system. Successive measurements data sets come in every 6 hours: $w^0(t_1)$, $w^0(t_2)$, $w^0(t_3)$,...,, although only at somewhat random spatial locations and subject to various uncertainties. Meanwhile, successive forecasts , $w^f(t_1)$, $w^f(t_2)$, $w^f(t_3)$,..., are computed using forecast models, from a given initial condition. Forecast vectors are defined on the regular gridpoints. These two streams of events are connected by the data assimilation process, though a statistical interpolation method such as Kalman filter[1] .

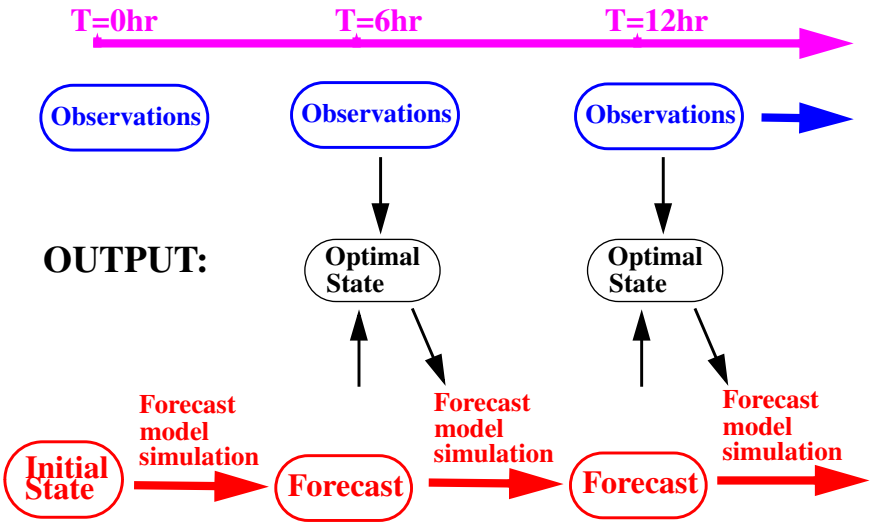


**Figure 1. Data Assimilation**. Data Assimilation connects the two otherwise unconnected streams of events.The output is the optimal states, sometimes called "analysis", which also serve as initial conditions for the next 6-hour period of forecast model simulations.

There are two major computational components in data assimilation, the advance of the weather state through weather/climate models and the statistical interpolation. The modeling part involves a set of issues different from the statisical interpolation; this part is typically described in a climate modeling context and will not be discussed further. We will instead focus on the statistical analysis part, given the measurement data $w^0(t)$ and the numerical model forecast $w^f(t)$.

Data assimilation process is a computationally intensive task: it is repeated every 6 hours continuously on an almost dedicated computer, consistent with the rate of the incoming observation data. This leads to the real-time requirement: a single assimilation cannot take longer than 6 hours. In fact, for a real-time forecast, everything including data assimilation, forecast simulation, other data gathering and manipulations/analysis, etc., must be done within 6 hours.

Furthermore, a complete research/study often requires that observation data sets be consistently assimilated going back perhaps to the 1980's , with the same model dynamics and covariance. However, both the climate models and covariance matrix are improved every few years now, and consistency requires assimilating observations all over again. The ambitious re-analyses (assimilation) of observation data of the past

decade relating to the Mission to Planet Earth Project would require a rate of 120 assimilations per day in real time. Similar re-analysis are being investigated at National Centers for Environmental Predictions and at European Center for Mediam-range Weather Forecast. These reanalyses require computing resources far exceeding those of traditional vector supercomputers.

The Physical-space Statistical Analysis System (PSAS) developed at the Data Assimilation Office (DAO) at NASA's Goddard Space Flight Center[2,3] is an advanced system which provides a general framework to perform the above data assimilation tasks. This software system will play a central role in NASA's Mission to Planet Earth enterprise and is designated to replace the existing operational system at the DAO in1998.

The computing resources required by this data assimilation go beyond the most up-to-date vector computers. In fact, a brute-force application of PSAS for a complete analysis requires about 4-7 hrs on a single head of a Cray C90. Recently, we have implemented all major parts of the PSAS package on massively parallel distributed-memory supercomputers Intel Paragon, Cray T3D and T3E, using message passing. We designed and incorporated several new algorithms which are highly efficient and scale well to large numbers (thousand) of processors. For example, a time-critical part of the package is the sparse linear equation solver, which achieves a sustained speed of 33.8 GFLOPS on a 512-node T3E-900 in 64-bit precision and 18.3 GFLOPS on a 512-node Paragon in 32-bit precision. As a result, the parallel PSAS package solves an 80,000-observation real problem in just 24.6 seconds (wall clock time, including everything from data I/O to partition of the unstructured grids) on the 512-node T3E-900. In contrast, the same problem takes estimated 5 hrs of CPU time on one-head of a Cray C90.

This paper describes new results obtained on the T3D and T3E together with earlier results on Paragon. Earlier results on Paragon have been briefly reported in [4].

## 2. The PSAS Algorithm

The purpose of the statistical analysis is to compute the optimal estimate $w^{opt}$ given the observation data $w^o$ and the numerical forecast $w^f$ (see Fig.1). The analysis is a challenging computational problem because the data sizes involved are large and the domain is unstructured.

The number of observations $N_0$ is around $10^5$ or more for operational systems. This is represented as a single vector $w^0$ . Note that all observation variables are treated equally, even though some of them, such as an east-west wind velocity data and a north-south wind velocity data, are actually taken at the same space locations.

The forecast data and the optimal data are defined on the regular grid points. At $2^o$ x $2.5^o$ horizontal resolution, there are 13104 grid points at each vertical level. The smallest forecast model system contains 14 vertical levels for four upper-air components and 3 sea-level components; they add up to $N_f$ = (3+4x14) x13104 = 773,136 model variables. The operational system will have 18-22 vertical levels, which brings $N_f$ to $10^6$ or more. All of these variables are represented as a single vector $w^f$, similar to $w^0$.

To calculate the Kalman gain matrix K requires inverting a $N_0$ x $N_0$ matrix. This would require a computation of order $N_0{}^3 = 10^{15}$ operations, a computation outside of today's capabilities. Fortunately, calculating K is not necessary if we do not need to

compute the error for $w^{opt}$, which is ignored at present. Instead, the PSAS algorithm solves the following equation for the vector x defined at the observation locations:

$$(HP^fH^T+R)\ x = w^0 - Hw^f \qquad (1)$$

The symmetric $N_0$ x $N_0$ matrix $M = HP^fH^T+R$ is called the innovation matrix containing correlations between all data points. The covariance matrix $P^f$ describes correlations between same type and different type variables [2]; it is calculated using a large number of existing subroutines containing both physical and empirical correlations. Using a Conjugate Gradient method, this innovation equation can be solved with a computation of order $N_0^2$. Afterwards, the solution x is folded back from observation locations to the regular grid locations to obtain the final optimal state through the increment

$$\Delta w = w^{opt} - w^f = P^fH^Tx \qquad (2)$$

which represents the net effect to the forecast $w^f$ due to incorporating the observation data through the assimilation process. Note $P^fH^T$ is a $10^5$ x $10^6$ sparse matrix.

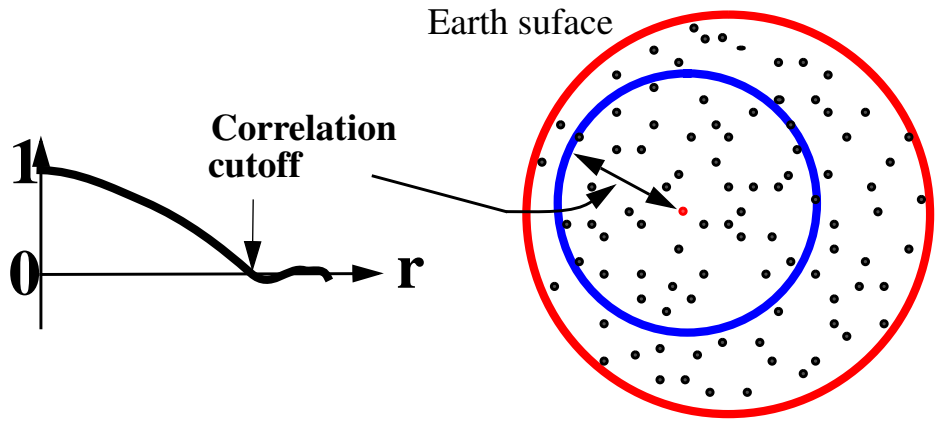


Fig.2. Correlation cutoff in original PSAS.

The critical part of PSAS is the solution of the innovation equation Eq. (1), with $10^5$ unknowns. The challenge of the problem lies in the size of the matrix involved and the unstructued nature. The matrix has a size of $10^5$x$10^5$ with 74% of the matrix entries being zero, due to the approximation of a cutoff in correlation of observations that are 6000km apart on the earth's surface (Fig.2). To store the entire matrix alone would require 20 GB (double precision) of memory, and it scales as $N_0^2$. It exceeds the capacities of any existing sequential computer. This difficulty is resolved in the Cray C90 codes by re-computing the matrix on the fly, at considerable expense of CPU time.

Correlation functions between two variables generally have a shape like a damped cosine[2] (see Fig.2). For computational convenience, they are cutoff at 6000km where they reach zero and the small oscillating tails are ignored (or replaced by a function which goes to zero smoothly). This makes the matrix $HP^f$ H a sparse matrix with about 26% of the matrix entries nonzero. The innovation matrix M has the same sparsity pattern.

This sparsity level makes conventional sparse matrix techniques inefficient, since they are typically used for matrices with nonzeros around 2% or less. In addition, conventional sparse matrix techniques achieve only the memory-bandwidth-limited processor speed due to the indirect indexing used there.

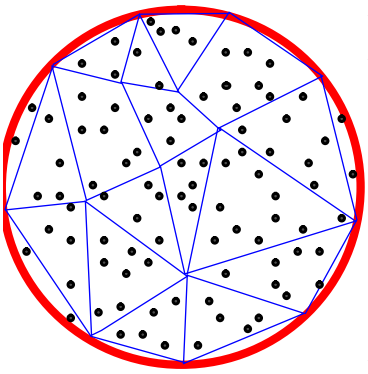

Fig.3 Regions of observations.

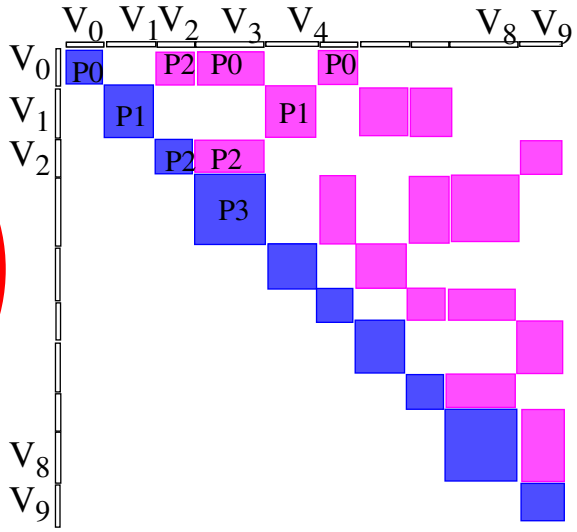


Fig.4 Block structure of the correlation matrix. Only nonzero blocks are shown. Due to symmetry, only half non-diagonal blocks are shown.

These problems are resolved by imposing a structural pattern to the sparse matrix M: observations are divided into regions with nearly equal numbers using a concurrent partitioner, and the correlation cutoff is enforced at the region level (see Fig.3). That is, all observations in one region are either correlated to all observations in another region or not at all, depending on whether the centers of the two regions are within 6000km. This slight modification on the physical correlations cutoffs leads to a correct and consistent solution, which differs from the original solution by 1-2% in an rms sense in our rigorous comparisons. However, this modification increases the calculation speed dramatically. The imposed structure is a block structure where 74% of the matrix blocks are identically zero (see Figure 4). Now the matrix-vector multiplication can be carried out using a level 2 BLAS routine, which is typically an order of magnitude faster than the memory bandwidth limited speed achieved by conventional sparse matrix techniques.

## 3. Solving the Innovation Equation

**Observations Partitioning**. After observations data are read in from disk and aritrarily distributed among processors, they are first grouped into regions of nearly same number, using a concurrent partitioner[5], which scales up as $\log_2 P$ on P processors. The partitioner uses a recursive inertial bisection algorithm, therefore leading to compact subdomains with good aspect ratios, which is best for this problem. Correlations among these regions result in the sparse block structure in the innovation matrix. Improvements have been made to the partitioner so that it now runs on an arbitrary number of processors,

instead of powers-of-2-number of processors only. Furthermore, the peculiar feature that many observation data (up to 42 in a case) are actually located on the same horizontal point (locations differ only in vertical levels) causes redundancies and ambiguity in the bisection algorithm. We modified the partitioner to always put these data into one group, resulting in slight variations in number of data in each final partition.

**Matrix Distribution**. Distribution of the huge innovation matrix proceeds first by calculating the sparsity pattern, which is represented by correlation lists identifying which observation regions are correlated to which other regions. Then a list of nonzero matrix blocks of this irregularly structured problem is generated. The large number of matrix blocks (e.g., there are 34907 matrix blocks in the 512-node case) must be distributed among the processors in a load-balanced way. This optimization problem on 34907 variables with various constraints is solved using a heuristic algorithm. After an initial trial distribution and several iterative improvements, much like a simulated-annealing, the algorithm finds a near-optimal distribution in just a few seconds. This step is carried out on a single processor while all the other processors are waiting. This sequential step costs about 5% of total time in the worst case of 512 processors.

The observation regions are then replicated among processors according to the matrix-block distribution list determined above. Due to the large correlation cutoff, each region correlates to about 1/3 of the total regions, i.e., each region is replicated on about 1/3 of total processors on average. There are large deviations to this rough average due to the irregular nature of the observation data distributions. This parallel step scales up roughly in proportion to total number of processors, about 3% of the total time in the worst case of 512 processors.

**Matrix Element Calculation**. Matrix blocks are calculated using the large number of Fortran routines describing both physical and empirical correlations. The calculation of matrix entries is quite expensive because the physics correlations require many complex operations and involves many branchings, and therefore runs at typical scalar processor speed. The parallel package now spends a significant amount of time (about 16%) on this part, while the other critical parts, especially the previous dominant solver part, are speeded up much more dramatically in the parallel implementation. This step is nearly perfectly parallel, which scales according to the number of matrix elements per processor, with a 10% level of load-imbalance inherent in the problem.

**PCG Solver.** The innovation equation is solved by a preconditioned conjugate-gradient iterative solver, of which the key part is the multiplication between the global matrix and the global vector[6]. Given the imposed matrix block structure, the multiplication proceeds similar to what one might called a parallel block approach for dense matrix-vector multiplication. In this parallel block approach, a dense matrix is divided into blocks which are then distributed; and a global vector is divided into sub-vectors which are then distributed/replicated. Added to this basic algorithm is a flexible structure to handle a block sparse matrix (in fact, 74% of matrix blocks are identically zero for this problem). The size of the matrix blocks varies from one to another, and the number of matrix blocks generally cannot be evenly divided by the number of processors. Furthermore, we only store the upper-right triangle matrix blocks due to symmetry; this allows each non-diagonal matrix block to be used twice in each matrix-vector multiplication, and therefore increases the computation/communication ratio. Communication here is

irregular for the sparse matrix; and storing only upper-right triangular half of the matrix adds more irregularity to the communication pattern. When everything is properly implemented, this new algorithm for the "not-so-sparse" sparse matrix-vector multiplication is highly efficient and scales well to a large number of processors, as indicated by the performance numbers shown here. We use a diagonal block preconditioner, which reduces the required number of CG iterations for a given accuracy by a factor of 2-3. The solution of the preconditioner equation is well-balanced and is carried out independently on each processor by solving one diagonal block at a time using a standard CG solver.

## 4. Folding back to Regular Grid

Folding back the solution to a regular grid at all vertical levels, i.e., calculating $\Delta x = P^f H^T x$, presents a very significant computational task, mainly due to assembling the $N_f$ x $N_0$ matrix $P^f H^T$. The matrix $P^f H^T$ has similar sparsity pattern as the innovation equation matrix M, except it is not symmetric. The folding-back process is dominated by assembling the huge matrix $P^f H^T$. Fortunately, the matrix-vector multiplication is carried out only once, so that the entire matrix does not need to be stored at the same time; matrix blocks are computed as needed on the fly, one matrix block after another.

Matrix $P^f H^T$ couples variables defined at observation locations to those defined on regular grid points. We make use of the fact that the observation locations have been grouped into regions and distributed among processors in a balanced way during the partitioning process. To efficiently implement the correlation cutoff at 6000km, the regular grid points have to be grouped into grid regions similar to grouping of observations for the innovation matrix, and then properly distributed among the processors at the beginning of the fold-back process.

**Grid Regions**. The 13,104 grid points on the $2^o$x$2.5^o$ mesh are grouped into 128 static rectangular regions based on latitudes and longitudes, which are different from the observation regions in shape, size and location. The entire surface of the globe is first divided into 11 zones, each of which is an $18^o$ strip between two latitudes specified in a file, except at north/south poles where a zone covers the entire circular area within $9^o$ latitude from the pole. The zone covering the equator is divided into 18 regions. Zones closer to the north/south poles are divided into fewer and fewer regions in a manner which gives them roughly the same area. The entire zones covering the north and south poles are single regions. Grid points on higher vertical levels are grouped similarly, such that a single grid region is a contiguous column from sea-level to the top of the atmosphere.

**Grid Thinning**. The number of grid points in each grid region varies substantially, even though all regions have similar surface area. For this reason, in those zones above $45^o$ latitude, grid points in the longitudinal direction are thinned gradually, so that the number of grid points remaining in each region become roughly same as the number of grid points in equatorial regions. The total number of grid points is reduced in this way to 8792. The matrix-vector multiplication will only use these remaining grid points, and values at the thinned grid points are obtained by interpolated from neighboring unthinned points. This reduces the total computational effort by 1 - 8792/13104 = 31%. The implementation keeps this thinning process as a run-time option so that one can check the consistency of the final solution.

**Grid Distribution**. Since observations are already distributed, we distribute grid regions according to the sparsity pattern of the $P^f H^T$ matrix. On each processor, we loop through all 128 grid regions; if a grid region correlates to (is within 6000km of) at least one of the observation regions on the processor, this grid region is retained on the processor. Since a given grid region may correlate to many observation regions on many different processors, grid regions may be copied to different processors. In fact, on average, a grid region is replicated on 1/3 of all processors.

**Folding Back**. The matrix-vector multiplication proceeds on each processor by going through all correlated pairs between a grid-region and an observation-region. For each such pair, the matrix block is first calculated. For upper-air components, the matrix blocks between the observation region and the grid region at all vertical levels are calculated at once. (In this way, we can make use of the specific form of the correlations to reduce significantly the amount of computation required at a later stage.) Then the multiplication between the matrix block and subvector is carried out and the result is accumulated using a single BLAS routine. They proceed independently and simultaneously on all processors.

Next we sum together increment sub-vectors on different processors to form the final results. Since the grid regions on each processor differ, the order of increment sub-vectors is different as well. So we reshuffle the sub-vectors on each processor into an universal vector which has identical order on all processors. In the universal vector, the components not present on a processor are set to zero. Afterwards, a global sum over the universal vector on all processors is performed. This final results is written to a binary file. In the case where grid points have been thinned, the universal vector components are reshuffled and the length of the increment vector for each component at each level is restored to the original 13104, with values on thinned grid points being simple interpolations between nearest unthinned grid points along the longitudinal direction.

## 5. Performance

The parallel PSAS implementation is completed on Intel Paragon and Cray T3D/T3E using MPI as the message passing system; minor differences in precisions (32-bit on Paragon and 64-bit on T3E/T3D) are accommodated by using C preprocessor #ifdef etc. The accuracy has been verified on smaller problems where the sequential results can be readily obtained. We carried out runs on various real problems on Paragon, T3D/T3E with increasing problem sizes on increasing number of processors. Table 1 summarizes the timings for a 79,938 observations problem with folding back to all 14 vertical levels on 512 processors. All timings are wall-clock time

In general, T3D is about twice faster than Paragon (note that on Paragon we use 32-bit precision and all Cray machines are 64-bit precision only). T3E is about 3.2 times faster than T3D, although the peak speed increases a factor of 6 (900 MFLOPS vs 150

MFLOPS). Most pure Fortran 77 codes get about 4x speedup or better. However, the time-critical level 2 BLAS SGEMV get only about 2.7x speedup.

TABLE 1. Timings (seconds) for an 79,938 observation problem on 512-node T3E, T3D and Paragon

| **T3E-900** | **T3D** | **Paragon** | **tasks** |
|---|---|---|---|
| 1.2 | 6.2 | 14.6 | read input data |
| 0.3 | 0.5 | 3.1 | partitioning observations |
| 1.1 | 1.7 | 3.8 | calculate nonzero blocks and distribution lists |
| 0.8 | 1.2 | 3.3 | replicate observation regions |
| 3.9 | 6 | 23.8 | calculate matrix entries |
| 10 | 26.7 | 36.4 | solve innovation equation using PCG solver |
| 0 | 0.8 | 1.7 | other miscellaneous parts |
| **17.3** | **42.6** | **72** | sub total for innovation equation |
| 6.9 | 35 | 67.6 | assemble PH matrix and multiply PHx |
| 0 | 0.1 | 0.4 | create grid regions, correlation lists |
| 0.1 | 0.4 | 1.5 | re-order and interpolate increment vector |
| **7.0** | **35.4** | **69.5** | sub total for folding back |
| 0.3 | 0.6 | 2 | write increment vector into disk |
| **24.6** | **78.6** | **158** | total time |

The identical tasks are also performed on the Cray C90 by J. Guo at the DAO using the sequential Cray codes. The CPU timings are listed in Table 2., along with corresponding MPP timings for comparison. Overall, the solution time is reduced by a factor of 740, from about 5 hours on the one-head C90 to 24.6 seconds on a 512-PE T3E. (Because of the expense of C90 time, the complete run for this large problem was not carried out. Instead, estimates were made based on runs on smaller problems.) This 740-fold reduction in solution time comes from two factors. One is the high Mflops achieved on the 512-PE T3E ( 34 GFLOPS vs. about 130 Mflops on 1-head C90). The other fact is that on C90 the innovation matrix blocks are re-computed on the fly as needed due to memory shortage, which therefore considerably increases the total computation unnecessarily.

TABLE 2. Solution time comparison. * indicates estimates. Timings on C90 are obtained by J. Guo at DAO.

| **Tasks** | **1-head Cray C90** | **512-PE T3E-900** | **512-PE T3D** | **512-node Paragon** |
|---|---|---|---|---|
| Read data and solve equation | 9120 | 17.3 | 42.6 | 87 |
| Fold back and write data | 9120* | 7.3 | 37.0 | 71 |
| Total | 18240* | 24.6 | 78.6 | 158 |

We analyzed the performance of the PCG solver on a 512-node Paragon in further detail (row 6 in Table 1). The solver achieves a sustained speed of 18.3 GFLOPS for an 85000 observation problem. (On 512-PE T3D and T3E, the sustained speed in the PCG solver is 12.6 and 33.8 GFLOPS respectively.) This represents 36% of the theoretical total peak speed of 51.2 GFLOPS. The solver spends 27.5% of the time on communications for this problem on 512-nodes. During each iteration, most of the communication time is spent on sending or receiving (on average) 536 messages per processor with (on average) 166 floating point numbers in a message. These numbers indicate a tightly coupled system like T3E is necessary for this problem.

## 6. Conclusions

We have designed and implemented a set of efficient and scalable algorithms for the PSAS data assimilation package, and achieved a 740-fold solution time reduction on a 512-PE T3E parallel platform over a single head of a Cray C90. This clearly demonstrates that data assimilation problems are well suited for distributed-memory massively parallel computer architectures. In particular, this work demonstrates that irregular and unstructured problems such as the data assimilation problem can be efficiently implemented on this type of architecture, with good understanding of the problem, careful (re)design of all necessary algorithms, and effective use of explicit message passing. By focusing on the algorithms, the application achieves a high sustained performance. Since no specialized optimizations targeted for the particular computer are done, these performance numbers are representative of what could be achieved by average users.

**Acknowledgment.** This work was funded by a NASA HPCC ESS Grand-Challenge Application, adminstered through University of Maryland (BG97-06100). This research also uses resources of the National Energy Research Scientific Computing Center, which is supported by the Office of Energy Research of the U.S. Department of Energy through contract number DE-AC03-76SF00098.

References

[1] R. Daley, “Atmospheric Data Analysis”, Cambridge University Press, New York, 1991.

[2] J. Pfaendtner, S. Bloom, D. Lamich, M. Seablom. M. Sienkiewicz, J.Stobie, A. da Silva, Documentation of the Goddard Earth Observing System Data Assimilation System - Version 1, NASA Tech Memo 104604, v.4, Goddard Space Flight Center, January 1995.

[3] S.E. Cohn, A. da Silva, J. Guo, M. Sienkiewicz, and D. Lamich, Assessing the Effects of Data Selection with DAO’s Physical-space Statistical Analysis System, Tech Report 97-08, Data Assimilation Office, Goddard Space Flight Center, April 1997.

[4] C.H.Q.Ding and R.D.Ferraro, “A Parallel Climate Data Assimilation Package”, SIAM News, November 1996, pp.1-11.

[5] C.H.Q. Ding and R.D. Ferraro, “Slices: A Scalable Concurrent Partitioner for Unstructured Finite Element Meshes”, Proceedings of 7th SIAM Conference for Parallel Processing, pp.492-493, 1995.

[6] C.H.Q. Ding and R.D. Ferraro, “A General Purpose Parallel Sparse Matrix Solvers Package”, Proceedings of 9th International Parallel Processing Symposium, pp.70-76, April 1995.