Climate Data Assimilation on a Massively Parallel Supercomputer

Chris H.Q. Ding and Robert D. Ferraro
Jet Propulsion Laboratory
California Institute of Technology
Pasadena, CA 91109

ABSTRACT. We have designed and implemented a set of highly efficient and highly scalable algorithms for an unstructured computational package, the PSAS data assimilation package, as demonstrated by detailed performance analysis of systematic runs on up to 512-nodes of an Intel Paragon. The preconditioned Conjugate Gradient solver achieves a sustained 18 Gflops performance. Consequently, we achieve an unprecedented 100 -fold reduction in time to solution on the Intel Paragon over a single head of a Cray C90. This not only exceeds the daily performance requirement of the Data Assimilation Office at NASA's Goddard Space Flight Center, but also makes it possible to explore much larger and challenging data assimilation problems which are unthinkable on a traditional computer platform such as the Cray C90.

1. Introduction

Data assimilation extracts maximum amount of information from available observations using climate models of Earth system. The assimilation transforms observations from diverse sources at arbitrary space and time locations into a regularly gridded, time-continuous data set. It propagates information into unobserved regions thereby complementing the observations.

The physically-consistent, time-continuous data set defined on regularly spaced grid points produced by the data assimilation has inherent usefulness in a broad range of applications. One of its primary use is to provide a direct confrontation between numerical model predictions with actual observations, i.e., incorporating the actual observations into numerical weather prediction and/or climate modeling. Since observational data come with errors due to various uncertainties, the data are incorporated in a statistical process much like a Kalman filter[1]. Because the observations are irregularly distributed on the surface of the earth (both latitude, longitude and elevation), and their positions change from time to time, the filtered observations must be interpolated to a regular grid on which model systems are based.

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 by 1998.

Currently, a brute-force application of PSAS for a complete analysis requires about 4-7 hrs on a single head of a Cray C90. This falls far short of the DAO real-time computational requirements in a production environment (see Sec. 2). Recently, we have implemented all major parts of the PSAS package on the Intel Paragon. We designed and incorporated several new algorithms which are highly efficient and scale well to very 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 18.3 GFLOPS on a 512-node Paragon (see Sec. 9). This represents 36% of the theoretical total peak speed of the 512-node Paragon.

As a result, the parallel PSAS package solves an 80,000 observation problem in just 158 seconds (wall clock time, including about 18 seconds spent on reading/writing data from/to disk) on the 512-node Paragon. In contrast, the same problem takes an estimated 5 hrs of CPU time on the Cray C90 (see Sec. 9). This represents an unprecedented 100+ -fold reduction in solution time. This parallel PSAS not only meets and substantially exceeds the real-time production requirements, it will in fact change the DAO operations significantly. Many problems previously unexplored due to their huge computational requirements now can be solved in a reasonable amount of time.

2. The Data Assimilation Problem

Let us first discuss the operational environment in which the data assimilation is carried out. Observations of the weather system come from ground stations, satellites, flying balloons, and many other sources. The collection of all these data within a given short time period indicate the state of the weather system, w_o. In a normal operational environment, successive observation data sets come in every 6 hours: w_o^1, w_o^2, w_o^3,..., giving us a record of what actually happened in the Earth surface system. Note that these observation data sets are not defined at exactly the same space locations from one set to the next (due to the movement of satellites, the rotation of the earth, and the irreproducibility of balloon tracks), and the number of observations differ from set to set. Furthermore, observations come with errors due to various uncertainties.

Meanwhile, successive forecasts of the state of the weather system, w_f^1, w_f^2, w_f^3,..., are computed using forecast models, from a given initial condition. Note that in most forecast models, the weather system on the surface of the Earth is represented by variables defined on a regular 2^ox2.5^o grid, with 14-22 elevation levels.

These two streams of events are connected or fused together by the data assimilation process. Given a forecast w_f which describes what the weather should be, and given the actual observations w_o which comes with uncertainties as reflected in the error covariance matrix R, the data assimilation process is to obtain an optimal estimate w_f^{opt} of the state of the weather system through a statistical process similar to a Kalman filter [2,3],

$$w_f^{opt} = w_f + K(w_0 - Hw_f) \qquad (1)$$

where H is an interpolation operator interpolating forecast variables from grid locations to observation locations, and

$$K = P^f H^T (HP^f H^T + R)^{-1} \qquad (2)$$

is the gain matrix. The matrix P contains complex physical correlations between same type and different type variables [2], calculated using a large number of existing routines.

This data assimilation accomplishes several important things. It produces the optimal data set combining observations with the numerical forecast. It transforms the observations from diverse sources at arbitrary space locations to the regular grid. This alleviates the difficulties associated with observations that change from time to time, both in number and in location, and propagates information from observed regions to unobserved regions.

The optimal physically-consistent, time-continuous data set defined on regularly spaced grid points produced by the data assimilation has inherent usefulness in a broad range of applications. In particular, it can be used to produce more accurate forecasts by using the assimilated data set as the initial condition in computing the next forecast, leading to an better forecast sequence, w_f^{1opt}, w_f^{2opt}, w_f^{3opt},... Furthermore, by examining the differences between the model forecast and the assimilated data (not the observations data), one may get hints and improve the forecast model itself to get a better forecast. (We emphasize that although the data assimilation relies on a model forecast as a correct criteria to assimilate to, forecast models are independent of the data assimilation.)

In operational environments, the data assimilation process is a computationally intensive task: it is repeated every $\Delta t = 6$ hours continuously on an almost dedicated computer, consistent with the periodicity of the incoming observation data. This leads to the real-time requirement: a single assimilation cannot take longer than the time period Δt. In fact, for a real-time forecast, everything add together including data assimilation, forecast simulation, other data gathering and manipulations/analysis, etc., must be done within Δt. Furthermore, a complete research/study often requires consistently assimilated data going back perhaps to the 80’s. In fact, the ambitious re-analysis (assimilation) of the past 5 years of data relating to the Mission to Planet Earth Project would require a rate of 120 assimilations per day in real time. The parallel PSAS package we have implemented, which reduces solution time from 5 hrs on C90 to 3 min on 512-node Paragon, now adequately meets this challenge.

3. The PSAS Algorithm

The PSAS problem is a challenging computational problem because the sizes involved are large. 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 final assimilated data are defined on the regular grid points. At $2^o \times 2.5^o$ horizontal resolution, there are 13104 grid points at each elevation level. The smallest forecast model system contains 14 elevation 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 elevation 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.

It is now clear that to calculate the gain matrix K requires inverting a N_0 x N_0 matrix $M=HP^fH^T+R$. This would require a computation of order $N_0^3 = 10^{15}$ operations, a computation outside of today's capabilities. Fortunately, calculating K is only necessary if we want to compute the error covariance matrix for the assimilated data (in Kalman filter theory, K determines how error covariances propagate in time), and they are ignored at present. Instead, the PSAS algorithm solves the following equation for the vector x defined at the observation locations:

$$(HP^fH^T+R)\ x = w_0 - Hw_f \tag{3}$$

The symmetric N_0 x N_0 matrix $M = HP^fH^T+R$ is called the innovation matrix. 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

$$\Delta w = w_f^{opt} - w_f = P^fH^Tx \tag{4}$$

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.

4. The Need for a Distributed-Memory Parallel Computer

The critical part of PSAS is the solution of a large linear system of equations, the innovation equation Eq. (3), with 10^5 unknowns. The challenge of the problem lies in the size of the matrix involved. The matrix has a size of 10^5x10^5 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. To store the entire matrix would require 10 GB (in single precision, or 20 GB in double precision) of memory, exceeding 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.

The memory-bound problem fits well into distributed-memory parallel architectures, which can have very large total memory. For example, the 512-node Intel Paragon parallel computer at Caltech has a total 512 x 32MB=16GB. Furthermore, the huge amount of floating point computations required for the problem can be distributed to individual processors, reducing the problem solution time dramatically. In fact, our implementation of the PSAS problem reduced the solution time from 5 hr on a Cray C90 to about 3 min on 512-node Intel Paragon.

5. Reformulating Matrix Sparsity Patterns

Correlation functions between two varibles generally have a shape like a damped cosine. For computational convenience, they are cutoff at 6000km where they reach zero and the small oscilating tails are simply ignored. 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.

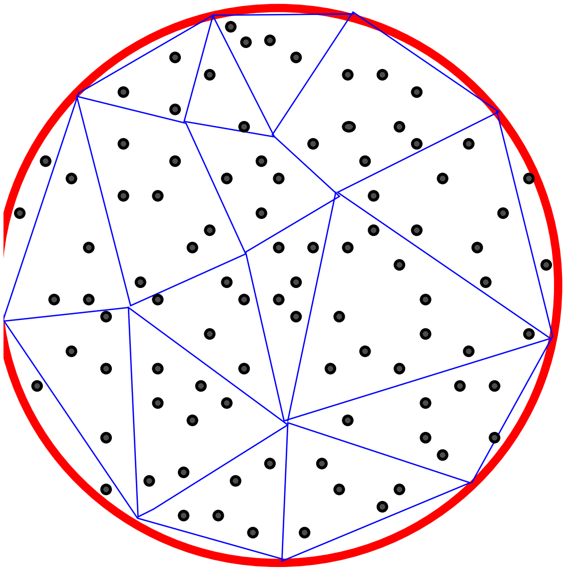

Fig.1 Regions of observations.

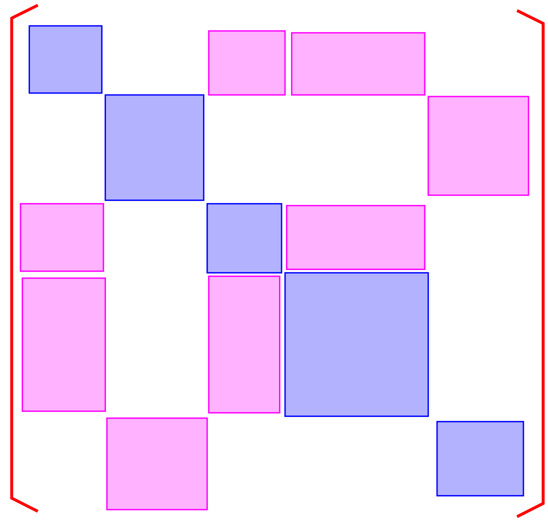

Fig.2 Block structure of the correlation matrix. Only nonzero blocks are shown.

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 Figure 1). 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 2). 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. For example, our solver runs at 77MFLOPS on 1 node of the Paragon, in contrast to about 10MFLOPS for conventional sparse matrix techniques.

6. Modular Programming Approach

Since observations are taken at irregularly spaced locations and they change from time to time, the PSAS problem is an unstructured problem whose parallel implementation is generally more complicated than regular problems. For this reason, we have developed a number of tools for unstructured problems, including an observations partitioner, a matrix-block distributor and a preconditioned Conjugate Gradient (PCG) solver.

We take a modular approach in programming structures. We carefully grouped high level data organizations, parallel implementation related parts, such as partitioner, solvers, etc., into individual modules, using structures in the C programming language. They are clearly separated from the lower level underlying physics details: the large number of correlations between different components, such as winds or water vapor mixing ratios, are rather complex (about 7500 lines of Fortran code spread over 40 subroutines). Interfaces to low level routines are written to restrict accesses through only the matrix block assembly and one or two C structures. The parallel codes contains 15,000 lines C codes and the 7500 lines existing Fortran codes; the original C90 Fortran codes has about 22,000 lines.

This approach has a number of benefits. It facilitates application development by restricting the physics part to a small section (albeit a lot of code for the complex correlations) so that the application scientists can easily modify it for various experiments. These routines use familiar Fortran and are free from considerations on how parallelism is achieved. It also maintains efficiency by using Fortran for numerical computations, and by using the highly efficient BLAS routines. For example, the PCG solver is essentially a skeleton code calling underlying Fortran routines and BLAS routines for numerical calculations. More importantly, this approach makes the high level data organization easier to develop and maintain. In fact, while we are developing the parallel implementation, the scientists at the DAO are developing new physical correlations/mechanisms, which will be easily incorporated into the parallel package later on.

7. Solving the Innovation Equation

Observation Partitioner. After observations are read in from disk and aritrarily distributted among processors, they are first grouped into regions of nearly same number, using a concurrent partitioner we have previously developed[4]. The partitioner uses an recursive inertial bisection algorithm, therefore leading to good aspect ratios for the resulting partitions which is best for this problem. Correlations among these regions results 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. Further more, the peculiar feature that many observation data (up to 42 in a case) are actually located on the same horizontal point (varies in elevation and in data type) cause 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 Distributor. Distribution of the huge innovation matrix proceeds first by calculating 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.

The observation regions are then replicated among processors according to the matrix-block distribution list and the matrix blocks are calculated using the large number of Fortran routines describing the exact physical 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 10-20%) on this part, in the light that the other critical parts, especially the previous dominant solver part, are speeded up much more dramatically in the parallel implementation (see Section 9).

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[5]. 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 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.

8. Folding back to Regular Grid

Folding back the solution to a regular grid at all elevation levels, i.e., calculating $\Delta w = P^f H^T x$, represents 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 any more. 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; they 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^ox2.5^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. For zones closer to the north/south poles, they 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 elevation levels are grouped similarly, such that a single grid region looks like a column sticking out from sea-level and reaching up to the upper 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^fH^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 elevation 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.

9. Performance

The parallel PSAS package is complete, and its accuracy has been verified on smaller problems where the sequential results can be readily obtained. We carried out runs on various real problems on the Intel Paragon 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 elevation levels on a 512-node Intel Paragon. The time on Paragon is wall-clock time.

TABLE 1. Timings for an 79,938 observation problem on 512-node Paragon

time (sec)	tasks
14.6	read input data
3.1	partitioning observations
3.8	calculate nonzero blocks and their distribution lists
3.3	replicate observation regions
23.8	calculate matrix entries
36.4	solve innovation equation using PCG solver
1.7	other miscellaneous parts
	sub total for innovation equation: 72
67.6	assemble PH matrix and multiply PHx
0.4	create grid regions, correlation lists
1.5	re-order and interpolate increment vector
	sub total for folding back: 69.5
2	write increment vector into disk
158	total time

From the table, one can see that solving the innovation equation costs 72 seconds, which includes 36 seconds spent on the PCG solver itself, 24 seconds on matrix entries assembly, and about 12 sec on the parallel overhead such as partitioning observations into 512 regions and generating load-balanced distribution lists for the correlation matrix.

The fold back now costs 70 seconds, which includes 0.4 sec spent on generating grid regions and distributing/replicating matrix block lists, 68 sec on assembling the matrix entries and carrying out the local matrix-vector multiplication, and 1.5 sec on global summing of all increment vectors, restoring vector length/order and interpolating to obtain values on thinned grid points.

The same 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 Paragon timings for comparison. Overall, the solution time is reduced by a factor of 115, from about 5 hours on the C90 to 3 minutes on a 512-node Paragon. (Because of the expense of C90 time, the complete

TABLE 2. Timing Comparison with Cray C90. * indicates estimates. Timings on C90 are obtained by J. Guo at DAO.

Tasks	1-head Cray C90	512-node Intel Paragon
Read data and solve equation	9120	87

TABLE 2. Timing Comparison with Cray C90. * indicates estimates. Timings on C90 are obtained by J. Guo at DAO.

Tasks	1-head Cray C90	512-node Intel Paragon
Fold back and write data	9120*	71
Total	18240*	158

run for this large problem was not carried out. Instead, estimates were made based on runs on smaller problems.) This 100+ -fold reduction in solution time is unprecedented in the area of high performance computing.

We analyzed the performance of the PCG solver in further detail, because our parallel algorithm design is primarily focused on it. The solver achieves a sustained speed of 18.3 GFLOPS on a 512-node Paragon for an 85000 observation problem (see Figure 3.). This represents 36% of the theoretical total peak speed of 51.2 GFLOPS. The solver achieves 77MFLOPS (77% peak speed) on one node. The reduction of efficiency on 512 nodes is due to various factors, such as communication, load-imbalance, etc. These numbers indicate the high efficiency the package has achieved. (The use o the level 2 BLAS routine for matrix-vector operations is crucial to achieve the high efficiency. On one Paragon node, the BLAS matrix-vector multiplication SGEMV gets 83 MFLOPS for matrix size of 1000^2 and 67MFLOPS for matrix size of 200^2.)

The PCG 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 percentage numbers on peak total speed and on communication indicate the highly scalable nature of the underlying algorithms.

It should be noted this performance is achieved under **standard operating environment** supported by vendor and involving **no special optimizations** targeted for the specific computer --- the performance is achieved mainly due to proper algorithm design/implementations, and therefore more representative of what could be achieved by average users. [The codes run under NX, not the much faster SUNMOS, and with no special assembly programming, except the standard vendor-supported BLAS package.]

Although Mflops rating is important for performance characterizations, problem solution time is the direct target for application scientists. The parallel codes reduces the entire problem solution time by more than a factor of 100 by going to 512-node Paragon from 1-head C90. We note that this dramatic reduction comes from two factors. One is the high Mflops achieved on the 512-node Paragon (vs. about 130 Mflops on C90). The other is the fact that on C90 the innovation matrix blocks are recomputed on the fly as needed due to memory shortage, therefore considerably increases the total computation unnecessarily. Note also, the computation on Paragon is in single precision (limited by 32 MB/node memory on Paragon; with accumulation variables in double precision, it appears to be adequate), while its double precision on C90. They both converge to same residual.

We note that our problem solution time includes IO and mesh partition/matrix distribution times, unlike many other applications where the problem mesh partitioned and matrix distribution are done in a separate preprocessing and not included in their solution time. This is because our codes includes partitioner, distributor, etc., in the same run-time executable.

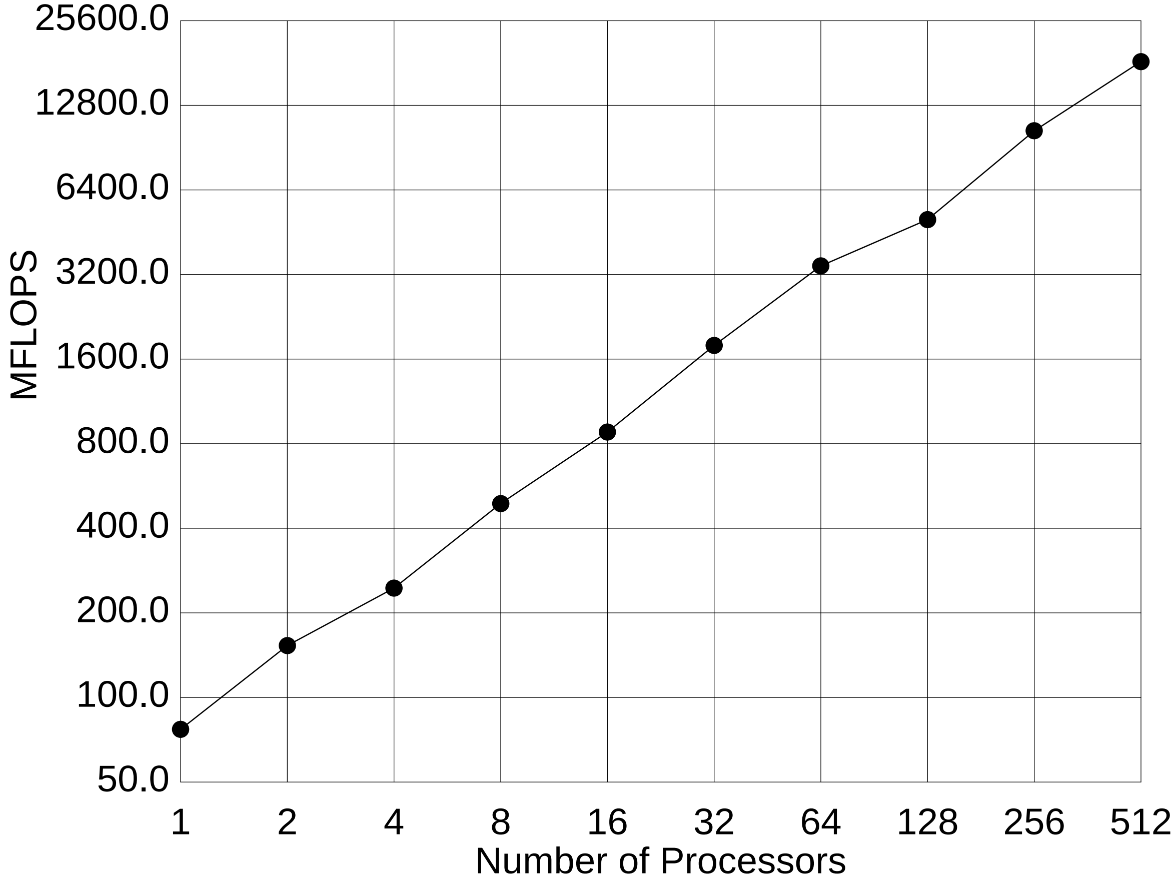

Figure 3. Performance of the PCG solver.

The two dominant parts remaining in the assimilation problem are the assembly of the matrix entries of the innovation and the fold-back matrix, as clearly indicated by their timings shown in Table 1. They are almost perfectly parallelized in our implementation, though a 10% level of load-imbalance still exists. This could be improved by using a more sophisticated distribution algorithm, thereby reducing the CPU time by about 10%. Because matrix entries are physical correlations which have many complex formulations, involving transcendental functions and many branchings, they are carried out at typical scalar processor speed (about 5GFLOPS in this case). Significant improvements in this part requires improving the sequential codes themselves. One direction is careful programming of the physical correlations to take full advantage of the RISC pipelined processor architecture. Another direction is to reformulate the basic computational stages such that some entities/combinations can be re-used many times, i.e., reduce the amount of computation itself. For example, calculation of the relatively expensive horizontal correlations should be structured in such a way that it is done once for all elevation levels. In fact, the programming structures in the fold back part have been designed with such an optimization in mind.

10. Conclusions

We have designed and implemented a set of highly efficient and highly scalable algorithms for the PSAS data assimilation package, and achieved a 100-fold solution time reduction on a 512 node Intel Paragon 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 involved 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 is done, these performance numbers are more representative of what could be achieved by average users.

The parallel PSAS package now meets and exceeds the DAO real-time production computing requirements. It will also improve the DAO operations significantly: many problems previously unexplored due to their huge computational requirements now can be solved in a timely manner.

Acknowledgment. This work is part of a NSAS HPCC ESS Grand-Challenge Application project. Clara Chan and Donald Gennery at JPL wrote portions of the code. We thank Peter Lyster, Arlindo Da Silva, and Jing Guo at the DAO for making the Cray C90 code available to us and helping us in understanding the problem. Access to the 512-node Paragon at Caltech is provided by the NASA HPCC Program.

The research described here was carried out by the Jet Propulsion Laboratory, California Institute of Technology, under a contract with the National Aeronautics and Space A.dministration.

References

[1] R. Daley, “Atmosphere 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] A. Da Silva, J. Pfaendtner, J. Guo, M. Sienkiewicz, and S.E. Cohn, Assessing the Effects of Data Selection with DAO’s Physical-space Statistical Analysis System, Proceedings of International Symposium on Assimilation of Observations, Tokyo, Japan, March, 1995.

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

[5] 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.