Appendix 1: Metacomputing and GEOS DAS-1 Assimilation System

Here the parallelization of an optimal interpolation algorithm is described as used in *Four Dimensional Data Assimilation*. The project has been conducted interdisciplinary at Syracuse University and NASA Goddard Space Flight Center. Combining the expertise in High Performance Computing by NPAC at Syracuse University and the expertise in atmospheric science by DAO provided a unique research environment.
Since the original algorithm is very complex a considerable amount of time was spend to ease the parallelization process. A parallel vector library in conjunction with a graphical user interface makes the definition of the parallel algorithm much more simple. The resulting program can be executed on top of the message passing library MPI. Experiments are conducted on an IBM SP2 distributed memory computer.
First, a small introduction to data assimilation and it's relationship to climate modeling and weather forecasting is given. A simplistic mathematical model is introduced to outline the major components of the 69000 line long optimal interpolation.
After being familiarized with the logic of the program, software engineering aspects are summarized which influence the strategy used for the parallelelization of the original sequential algorithm.
Next, a graphical software environment is described which is used to specify the parallel algorithm. The environment enables to specify parallel programs as process task graphs which are than mapped onto the parallel machine. During the execution of the program the resources of the parallel machine are monitored and dynamic load balancing is used to keep all parts of the machine as busy as possible.
In the last section results of experiments, conducted on a SP2, are given. Presented are results for two different versions of the OI code. The code uses data and task parallelism where appropriate. The final version of the algorithm scales very well with the number of processors. Suggestions for further improvement of the algorithm are given.

1 Climate Modeling and Data Assimilation

The precise prediction of weather and climate is an essential part of our daily life. The knowledge of future weather and climate conditions are used in decision processes of quite different scale. For example, while it is essential for a traveler to know the weather conditions of the place of destination for the current day or week, it is also important to know the climatological conditions in order to chose the appropriate season to start out a journey. Other important examples are the occurrence of smog and the influence of pollution on the climate of the earth.
Providing a precise forecast enables to determine proper preventive actions governed by the future atmospheric condition. Weather and climate models provide the opportunity to study these effects at different scales.
Figure 1 displays the space and time scale phenomena of the earth climate system[4]. The region of interest ranges from about 10km to 36000km, the radius of the earth. The time scale of interest ranges from a few hours to month, years, and for long running CO_2 analysis even longer.
Unfortunately, the atmosphere is the most variable component of an earth climate system. Elaborate models are necessary to describe the atmosphere[16, 17].
Not only do climate models contain complicated equations but perform its calculation on huge data sets. Thus, computers with enormous computational power are needed to calculate a prediction of only small complexity. Therefore, climate modeling is classified as one of the grand challenge problems. To motivate the role of data assimilation in relation ship to climate modeling a more detailed look into the field of meteorology is necessary.
The main problem in meteorology is how to obtain a valid forecast. Already, early in the development of the science of meteorology three steps are distinguished:

1. The (initial) state of the earth has to be determined.
2. Laws which predict the new state have to be determined.
3. The forecast is obtained while applying the Laws to the initial state.

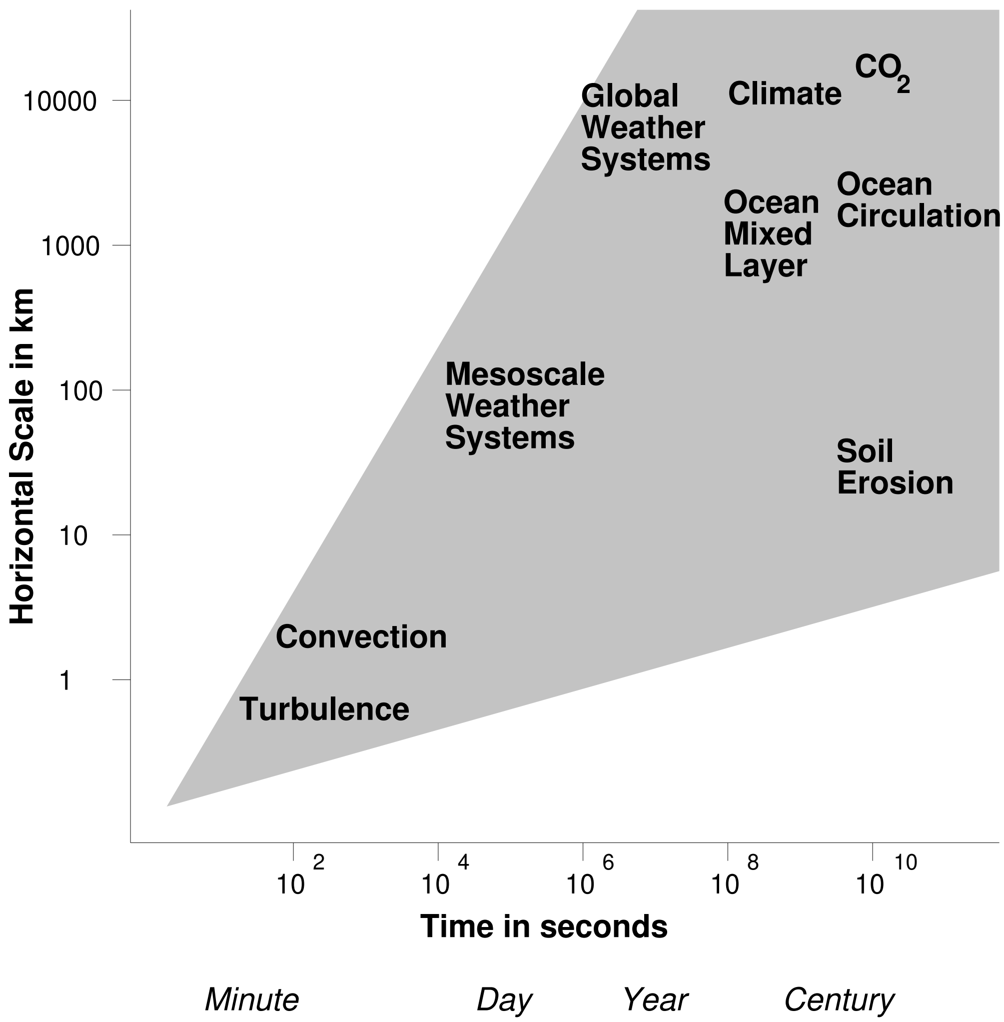

Figure 1: *Space and time scales of an earth climate system*

The three parts have a direct influence on each other. Without a precise initial state the forecast will be unreliable. In addition the governing equations describing the atmosphere should be as accurate as possible. Last but not least the application of a mathematical evaluation process of the governing equations applied to the initial conditions should introduce only a small error.
In the ground breaking publication by Richardson[12], a finite difference form is used to integrate forward in time, obtaining a forecast from an initial state. He used an initial state as published in [2]. The difference scheme is applied on a regular latitude-longitude grid of fixed size.
Different variables are used to describe the governing equations, e.g. pressure, height, the three components of the wind (often abbreviated as u-, v-, and z-component), and many more. Since these variables are defined for the whole domain, the term *field* is used for an array of variables of the same type.
Unfortunately, the initial data in Richardsons calculation did not define the values of the fields at all grid points of the domain (in today's forecast systems this is still the case). The forward integration can only be achieved after values for the fields have been defined. To quantify this problem, a typical distribution of observations is shown in Figure 2. A schematic closeup is shown in Figure 3. The question that arises, is how are the initial values for the computational grid determined?

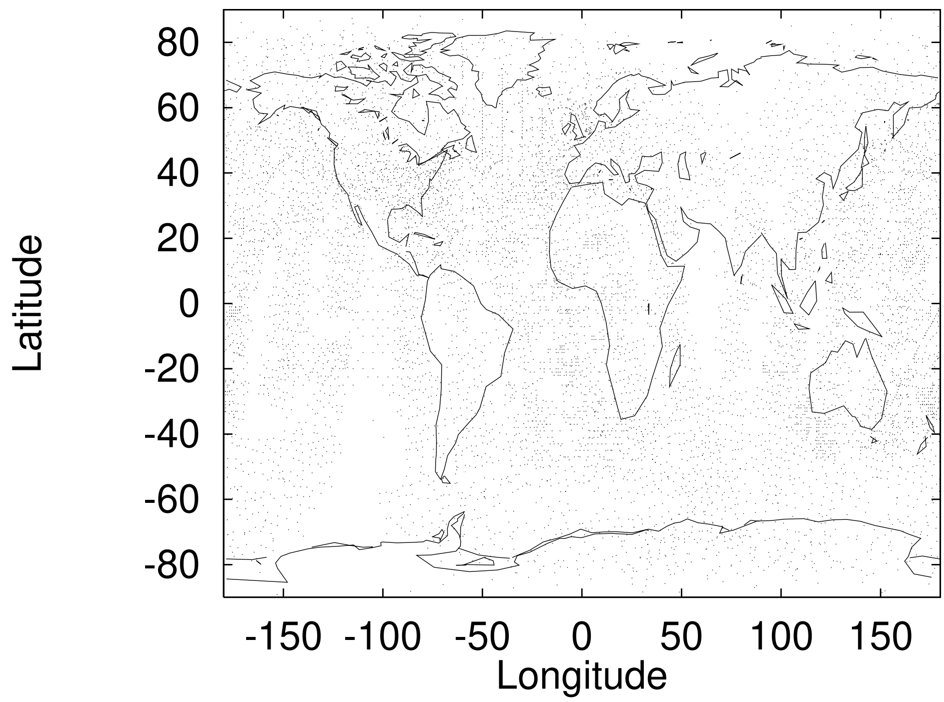

Figure 2: *Typical distribution of observations for the determination of an initial state*

Optimal Interpolation

Many possible solutions for obtaining the initial values have been introduced in literature. One of the most successful and still used methods, is the technique of *optimal interpolation*. It was introduced by Eliassen[5] and Gandin[6]. It uses a statistical process based on mean square minimization to obtain the missing values for the computational grid. The idea behind optimal interpolation is to take a first guess for the fields and observation. Then, for each gridpoint, actual observations are used in order to obtain an *analysis increment* based on the weighted sum of all observations in the vicinity of the point.
In order to obtain the first guess fields at the grid points, the model is integrated once forward in time. The first guess observations are determined by a interpolation step from the fields towards the actual observation locations.
In practice, a cut-off distance is used allowing only observations up to a fixed distance to be used for the update. This is shown schematically on the right side of Figure 3 (this strategy will be described in more detail later).
First, a simplified mathematical formulation of the optimal interpolation algorithm is given. This description follows the univariant case, which has the property that no correlation exists between the variables of one field and another. Let,

N_g be the number of observations, affecting a particular grid point g,

$\mathbf{A_g}$ be the resulting analysis at gridpoint g,

$\mathbf{F_g}$ be the first guess value at the gridpoint g,

$\mathbf{F_i}$ be the first guess value for the ith observation,

$\mathbf{O_i}$ be the ith observed value, and

$\mathbf{W_{gi}}$ be the yet undetermined weight function.

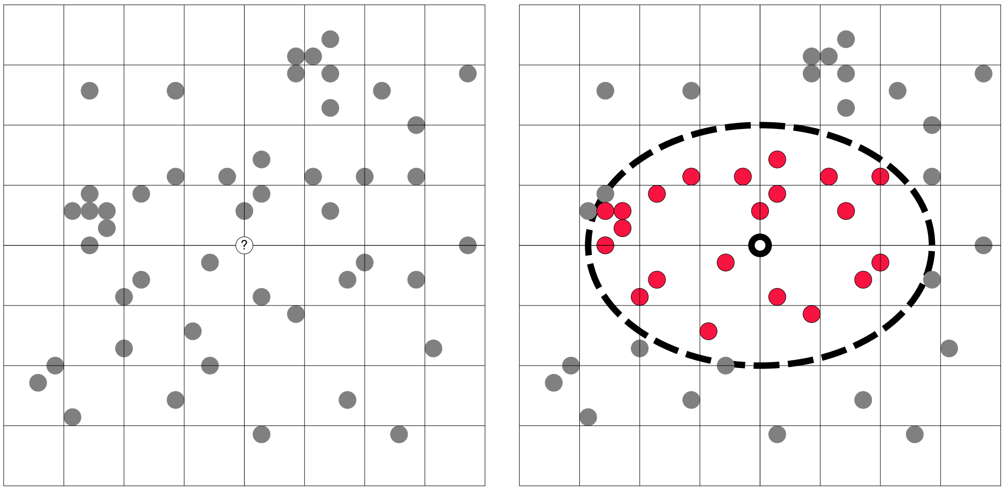

Figure 3: *Schematic closeup of a typical distribution of observations for the determination of an initial state. The area of influence is shown for the gridpoint in the middle)*

Let $\langle\ \rangle$ denote the statistical averaging process, and let $\langle \epsilon_g^{\mathbf{A}} (\epsilon_g^{\mathbf{A}})^{\mathbf{T}} \rangle = \langle \epsilon_g (\epsilon_g)^{\mathbf{T}} \rangle^{\mathbf{A}}$.
Then, the optimal interpolation algorithm can be derived as follows:

$$\mathbf{A}_g = \mathbf{F}_g + \sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{N}_g} \mathbf{W}_{g\mathbf{i}} (\mathbf{O}_\mathbf{i} - \mathbf{F}_\mathbf{i}) \tag{1}$$

Where $\mathbf{A}_g - \mathbf{F}_g$ specifies the *analysis increment* or *correction*, and $\mathbf{O}_\mathbf{i} - \mathbf{F}_\mathbf{i}$ specifies the *observation increment* or *innovation*.
Transforming Equation (1) in terms of errors yields to

$$\epsilon_g^{\mathbf{A}} = \epsilon_g^{\mathbf{F}} + \sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{N}_g} \mathbf{W}_{g\mathbf{i}} (\epsilon_\mathbf{i}^{\mathbf{O}} - \epsilon_\mathbf{i}^{\mathbf{F}}) \tag{2}$$

Then,

$$\langle \epsilon_g (\epsilon_g)^{\mathbf{T}} \rangle^{\mathbf{A}} \tag{3}$$

can be minimized with respect to the weights,

$$\frac{\partial}{\partial \mathbf{W}} \langle \epsilon_g (\epsilon_g)^{\mathbf{T}} \rangle^{\mathbf{A}} = \mathbf{0} \tag{4}$$

$$\langle \epsilon_g \epsilon_\mathbf{j}^{\mathbf{T}} \rangle^{\mathbf{F}} = \sum_{\mathbf{i}=\mathbf{1}}^{\mathbf{N}_g} \mathbf{W}_{g\mathbf{i}} (\langle \epsilon_\mathbf{i} \epsilon_\mathbf{j}^{\mathbf{T}} \rangle^{\mathbf{F}} + \langle \epsilon_\mathbf{i} \epsilon_\mathbf{j}^{\mathbf{T}} \rangle^{\mathbf{O}}) \tag{5}$$

In the Equation (5), correlations between the observations and the first guess errors are neglected. The errors are assumed to be uncorrelated and unbiased.
To complete the calculation, it is assumed that the forecast error covariances from the previous equation can be estimated by an empirical fit.
In the univariate case, the model and observed error correlation can be approximated as follows:

$$\langle \epsilon_\mathbf{i} \epsilon_\mathbf{j}^{\mathbf{T}} \rangle^{\mathbf{F}} \simeq \sigma_\mathbf{i}^{\mathbf{F}} \sigma_\mathbf{j}^{\mathbf{F}} \delta_{\mathbf{ij}}^{\mathbf{F}}$$

$$\langle \epsilon_\mathbf{i} \epsilon_\mathbf{j}^{\mathbf{T}} \rangle^{\mathbf{O}} \simeq (\sigma^{\mathbf{O}})^{\mathbf{2}} \rho_{\mathbf{ij}}^{\mathbf{O}}$$

where δ and ρ are functionals dependent on the observation positions, their difference, and the pressure.[1]
In the same way, let $\left\langle \epsilon_{\mathbf{g}} \epsilon_{\mathbf{j}}^{\mathbf{T}} \right\rangle^{\mathbf{F}}$, be approximated by

$$\left\langle \epsilon_{\mathbf{g}} \epsilon_{\mathbf{j}}^{\mathbf{T}} \right\rangle^{\mathbf{F}} \simeq \sigma_{\mathbf{g}}^{\mathbf{F}} \sigma_{\mathbf{i}}^{\mathbf{F}} \varrho(\mathbf{d})$$

Then, we obtain in a final form

$$\underbrace{\sigma_g^F \sigma_i^F \varrho(d)}_{b} \quad = \sum_{i=1}^{N} \mathbf{W_{gi}} \quad \underbrace{(\sigma_i^F \sigma_j^F \delta_{ij}^F + (\sigma^2)^O \rho_{ij}^O)}_{\mathcal{A}} \tag{6}$$

One can now solve this system of linear equations for each grid point in order to obtain the missing weights.

$$\mathcal{A}x = b \tag{7}$$

As mentioned earlier, the optimal interpolation process can also be used to allow observations of one kind of variable to influence the analysis of another one. This process is known as *multivariate analysis*. In the multivariate case, the correlation terms and the weights in the Equation (5) become matrices instead of vectors.
For a complete derivation of a multivariate optimal interpolation algorithm we refer to [4]. The parameter set used in the operational OI algorithm is described in [10].

Quality Control

Besides the use of the optimal interpolation strategy, the quality of the actual observation used to perform the analysis is of upmost importance.
A network of various observation instruments is used to gather data about the atmospheric condition. Unfortunately, a measurement taken by an instrument will contain errors. These errors have to be corrected.
Examples of different instruments are given below. They are categorized in three instrument classes.
Instruments of class 1, measure observations taken at a single point. Common examples of such instruments are thermometers, and devices measuring humidity. Class 1 instruments can even be placed in radiosonds to measure values at different height or pressure levels.
Instruments of class 2, sample an area or volume rather than an observation point. Common examples are radar measuring precipitation and winds via Doppler shift.
Class 3 instruments determine wind velocities from Lagrangian trajectories. Here, a physical target is followed remotely and the velocities are determined with the help of the displacement of the target. Examples for such instruments are radiosond balloons, but also cloud elements tracked with pattern recognition techniques from geostationary satellites.
Each of the instruments have different characteristics introducing errors into the observed value or variable. The characteristics are predetermined, and on average, known for each instrument type. They are an important input in the actual data analysis, in order to eliminate errors leading to big variations of the actual atmospheric state. Therefore, it is necessary to check the input data and correct or reject erroneous data from the analysis. Certain errors can be corrected quite simply leading to an overall improved quality of the analysis.

Operational Optimal Interpolation Algorithm

In this section, the specific details applying to the operational Optimal Interpolation Algorithm, as run at the NASA Data Assimilation Office (DAO), at Goddard Space Flight Center (GSFC), are described.
The NASA Four Dimensional Data Assimilation System is based on independent program modules (shown in Figure 4). First, data observed by satellites, weather balloons, airplanes, and other sources are prepared for input. A quality control check is performed to eliminate wrong or erroneous data. After the quality control, the model calculation is performed[11]. A general circulation model (GCM) – henceforth called *the model* – is used to generate a six hour forecast. The objective analysis – henceforth called *the analysis* – then involves

[1] An exact specification of these complex functions and their derivation can be found in [10, 4, 1].

use of statistical weights to combine the model grid-point data and the observations to obtain a best estimate for the state of the atmosphere.

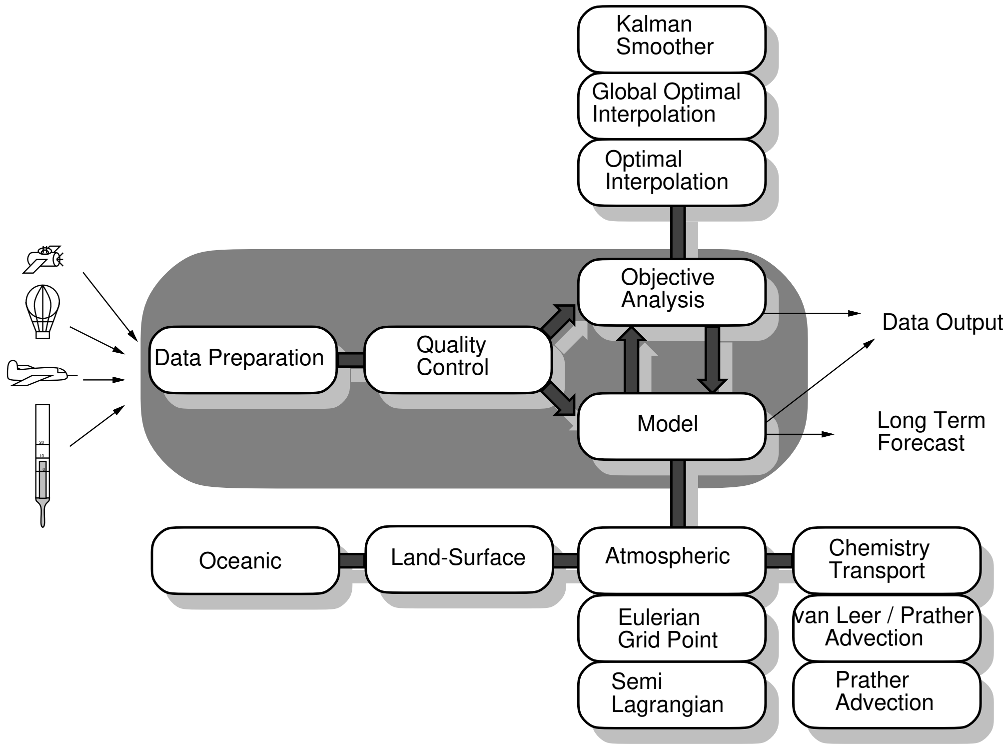

Figure 4: *Overview of the Integrated NASA Goddard Data Assimilation System.*

At the DAO, different strategies for the analysis are in use, as well as in development. For example a multivariate optimal interpolation algorithm[14, 1, 4], a global analysis algorithm, called the Physical-space Statistical Analysis System (PSAS)[15, 8], and the Kalman Filter[3, 9], are among the data assimilation strategies.
Even though the functional decomposition of the Goddard assimilation system (i.e. model forecast followed by analysis) may permit that different modules are executed on different machines. For the reminder of the report we assume that the different parts are executed on the same machine.
The analysis incorporates a variety of observations such as rawinsond reports, surface ship observations, and satellite retrievals. At present, one six hourly analysis incorporates approximately 100,000 observations. In ten years the number of observations is expected to increase in at least two orders of magnitude. Data are interpolated from non-uniform locations to a regular grid via the multivariate optimum interpolation (OI) analysis technique as introduced before[11]. The OI algorithm uses statistical estimates to determine appropriate relative weighting between *noisy* observations and a somewhat inaccurate first guess which is obtained with a forecast by the model. This is done in order to minimize the resulting error in the analysis. The analysis cycle consists of performing

1. the initialization,

2. the model forecast, and

3. the data analysis.

These steps are iterated for every 6 hour interval as shown in Figure 5.
The most important difference between the optimal interpolation algorithm specified in section 1 and the operational system is the use of so called *mini-volumes*. In the original formulation of the algorithm the

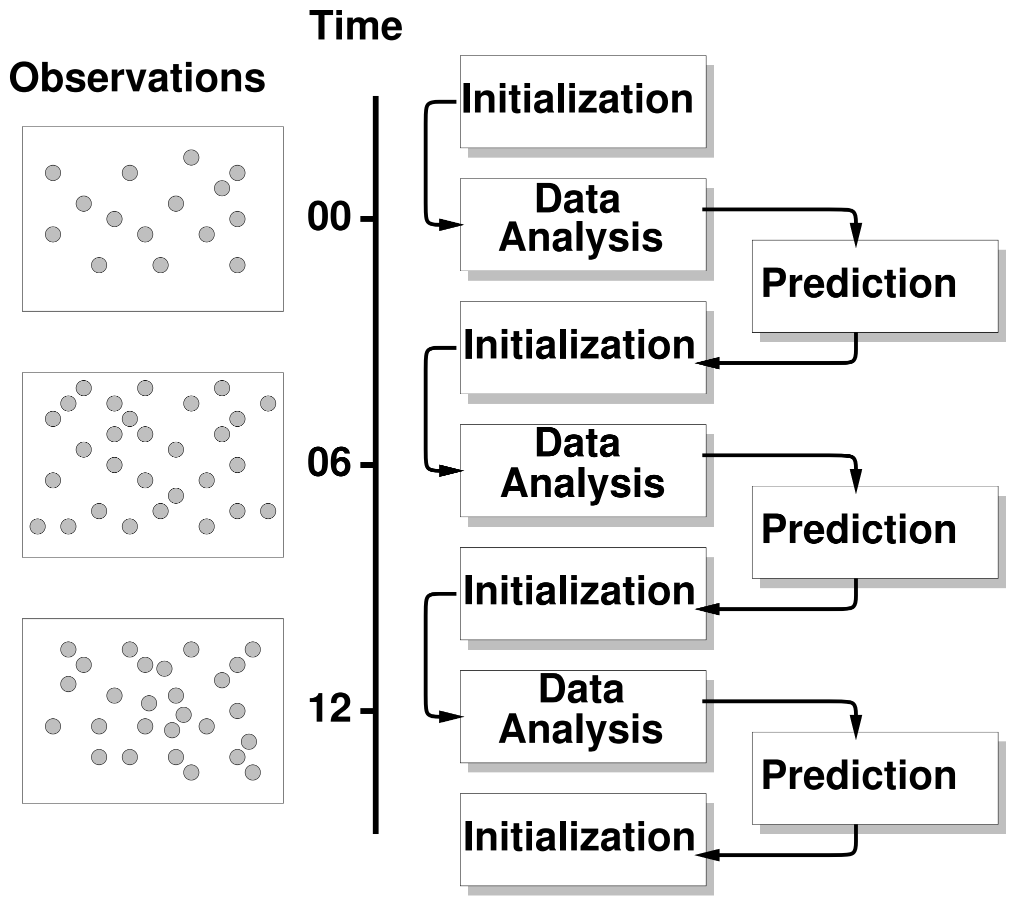

Figure 5: *Data Assimilation Cycle*

covariance matrices are computed for each grid point. Since the difference in the covariance matrices of one grid point and a neighboring gridpoints is rather small, the covariance matrices are determined for a group of grid points, laying in a small latitude-longitude-height box. Hence the term *mini-volume optimal interpolation algorithm* is used to describe this algorithm. The introduction of mini-volumes increases the calculation speed drastically but reduces slightly the quality of the solution in contrast to the original algorithm.
Parallel computing will allow to eliminate the mini-volume concept and perform the optimal interpolation on each grid point dependent on the computational resources available.
An important issue in climate modeling is the resolution as used in the grid representation. Is the grid dimensioned wrong, an error based on its representation is introduced, called *error of representatives*. If the scale of the grid is to big effects such as the well known *lake effect snow* in Syracuse would not be detected. Therefore it is desirable to have a model which contains a high number of grid points in a volume over the earth. It can be concluded that grids with big gaps between the gridpoints introduce a high error in the calculation. On the other hand the calculation performed on a computer will be much less time consuming.
The resolution of the grid can be chosen arbitrarily in the production code. Usually resolutions of 2x2.5x20, 2x2.5x46, or 4x5x20 are used.
Furthermore, the actual data assimilation procedure is split in three separate modules. Differentiated are the surface-level-pressure analysis(SLP), the moisture-vapor analysis(MIX) and the height-u-wind-v-wind analysis(HUV). The logical structure of the separate parts are similar. They distinguish each other through different parameters and equations describing the atmospheric condition, which is analyzed in each part. For simplicity we describe the functionality of the algorithm on an abstract level. code.
The following high level program segment describes the program flow of the quality control of the observation

data.

```
1  proc Quality Control
2         Read in the observations.
3         Ignore all observation with invalid
4           locations.
5         Obtain the first guess values at the
6           location of the observations.
7         Interpolate the forecast errors to
8           the observation locations.
9         call Gross Check.
10        call Buddy Check.
11 end proc
```

In line 9 and 10 interpolation algorithms are used to obtain the appropriate values at the location of the observation. Presently the quality control algorithm consists of two further steps: the gross-check and the buddy-check.

The *gross-check*, eliminates obvious errors, dependent only on the values at the location of the observation and performs a test based on the observation and forecast error variances:

$$\delta^2 > T_h((\sigma^O)^2 + (\sigma^F)^2) \tag{8}$$

where, δ is the difference between an observation and the interpolated background first guess value, and T_h is a subjectively defined tolerance value varying with height. The high level code of the gross check looks as follows:

```
T_h := Tolerance
proc Gross Check
     foreach observation ∈ this processor
        if pressure is in valid range
           then observation is valid
        fi
        if observation is still valid
           then
                 δ = variable^O − variable^F
                 if suspect
                    then fail
                    else if δ^2 > T_h((σ^O)^2 + (σ^F)^2)
                            then fail
                         fi
                 fi
           end foreach
        end proc
```

The *buddy-check*, eliminates errors which depend on all the values of all the observations in the vicinity of the location. A successive-correction method performs the analysis at each location which is marked as suspicious observation. Than, the difference between the analyzed value and δ are subjected to Equation (8). The observation is accepted if Equation (8) is not satisfied[14].

```
proc Buddy Check
    foreach valid observation ∈ this processor
        Search the observations in a particular
            radius surrounding the data point.
        if data is in tolerance level
            then reaccept
        fi
    end foreach
end proc
```

After the quality control is conducted the optimum interpolation algorithm is applied and the matrices, as given in Equation 7, are solved as follows:

```
proc Solve
    foreach Mini-volume ∈ this processor
        Collect the data for this mini-volume
        foreach Vertical level
            Calculate covariance matrixA
            Solve the linear equationAx = b
            Store the result
        end foreach
    end foreach
end proc
```

New values for the fields are stored and the algorithm is terminated when the equations for all minivolumes have been solved. The optimal interpolation is at this point completed.
The resulting fields will used as input to a prediction algorithm to determin the next state of the variables.

2 Software Engineering Aspects

While parallelizing the existing code the following project limitations had to be fulfilled: First, the program should run on as many MIMD machines as possible. Second, the language in which the mathematical computations are performed is Fortran as a natural extension of the present code and due to its performance advantage over other languages.
One major practical problem in the parallelization was the internal structure and software quality of the existing code. The program is written in Fortran and designed for a specific vector supercomputer. Because of its long development time and the large number of programmers involved in its development it is very complex and can be concidered as *legacy code*.
Unfortunately, the code does not pass the ANSI Fortran 77 standard. Therefore, it was necessary to use Fortran 90 as programming language of choice. This has the advantage that vector arithmetic operations which are part of the Fortran 90 programming language could be used in order to increase the readability of the program.
Another major problem was the development stage of the sequantial code. The first version of the code was buggy and while porting the sequential code with parallel extensions on different MIMD machines many bugs in the original program could be revealed. At first we tried to reimplement major parts of the algorithm in Fortran 90, but had to stop this effort because of time limitations bound to the project.
A further problem with the old version was the data input and output, which were based on ASCII file IO. Also the IO routines in the quality control were different from the production code.
The time spend in the parallelization of the code was only 10% of the total programming time, while the rest of the time was spend in optimizing the sequential code.
As result of this optimization the execution time of the sequential code could be improved by 50%. This shows once more that eliminating or improving inefficient code in a sequential algorithm can drastically improve the runtime. For a parallel code this meant also that the program could run on half of the processors using the same wall clock time.

In a separate effort, at NASA GSFC, a new version of the code was in development. This version of the code eliminated the known bugs in the previous version. The quality of the output and the execution could be improved, while incooperating parts of the changes done to the first version during it's parallelization. In addition, the matrix solve routines have been replaced by a matrix solver which is available in optimized form for the used hardware platforms.
Due to the availability of the new version, which behaved much more stable and had increased performance, we decided at the end of November 95 to abandon the code development of the version 1.2. Instead we took the new version and incooperated changes to parallelize the code.
Nevertheless, we would like to point out that the new sequential code can still be improved by another 30% based on the inefficiencies we discovered in the previous version.
Before, performance results are given the software engineering aspects of the code development are summerized.

Software Engineering Implications

The parallelization of the existing program is a difficult task. In order to minimize the parallelization effort we developed a strategy to transform the optimal interpolation algorithm gradually into a parallel program[19]. To be portable on the next generation of MIMD supercomputers, only widely supported programming languages and tools are used. The choice of programming languages and software tools are based on their availability, their ease of use, their flexibility, and their standardization.

Fortran 90 and HPF: For the computational core of the main program Fortran 90 is used. In addition it will be possible to transfer major parts of the Fortran 90 OI program to HPF once the compilers are more stable.

C/C++: For some of our load balancing and data redistribution tools C/C++ is used, because data abstraction and the development of software libraries are easier in an object oriented programming language. We believe that the incorporation of mathematical data structures in the language (as suggested by the ANSI C++ committee) will enable easier combination of Fortran and C++. For the parallel algorithm no performance loss is expected. This is due to the fact that the C/C++ routines are only used for intercommunication procedures. Interfaces to Fortran are provided. Nevertheless, most of the libraries exist also in a Fortran90 to fulfill the requirement of a pure Fortran program.

LAPACK: Where possible the LAPACK and BLAS libraries are used because for a particular machine fast optimized versions of the libraries exist. This boosts the performance of the algorithm drastically.

Message Passing: As intercommunication Library the standard Message Passing Interface *MPI* is used. The routines used for message passing are chosen to be as simple as possible so that a replacement of the underlying message passing library is easy. Therefore, a port to PVM is easily possible while replacing the elementary message passing calls. Due to the design and extensions of the used message passing libraries the usage of heterogeneous computing environments is also possible. MPI will provide an efficient interface to supercomputers.

Portability: Due to the designed strategy, introduced here, a highly portable program can be designed running on the current and next generation of supercomputers.

Code Development: In addition, a Graphical Software Environment is used to simplify and support the parallel code development.

3 Graphical Software Environment

The experience gained from parallelizing the first version of the code and the interaction with the atmospheric scientist reviled, that there is a need to simplify the parallelization of the OI code and other codes in atmospheric science. Therefore, a visual software environment is developed to ease the process of new sequential and parallel program development[7]. The programming environment forces the programmer to use a more strict and regulated method of programming making the code ultimately more easy to maintain and avoid problems existing in egacy codes. The environment allows one to specify a parallel program in a visual graph editor and fill in the details in an arbitrary programming language, which in turn are than compilabled on a set of specified the target machines.
This environmnet is not only useful for the development of the parallel OI code, but also for the development of many other parallel programs. Several steps for the development of a parallel program are distinguished:

1. Designing of the global program structure, keeping in mind the parallel nature of the problem.
2. Specifying the functionality of the blocks which build the global program structure.
3. Conversion of program blocks which are of concurrent nature into parallel blocks.
4. Mapping the parallel and sequential blocks on a real architecture.
5. Running the parallel program and observing performance statistics.

In Figures 6- 9 snapshots of the software environment are displayed, as used for the parallel optimal interpolation code.

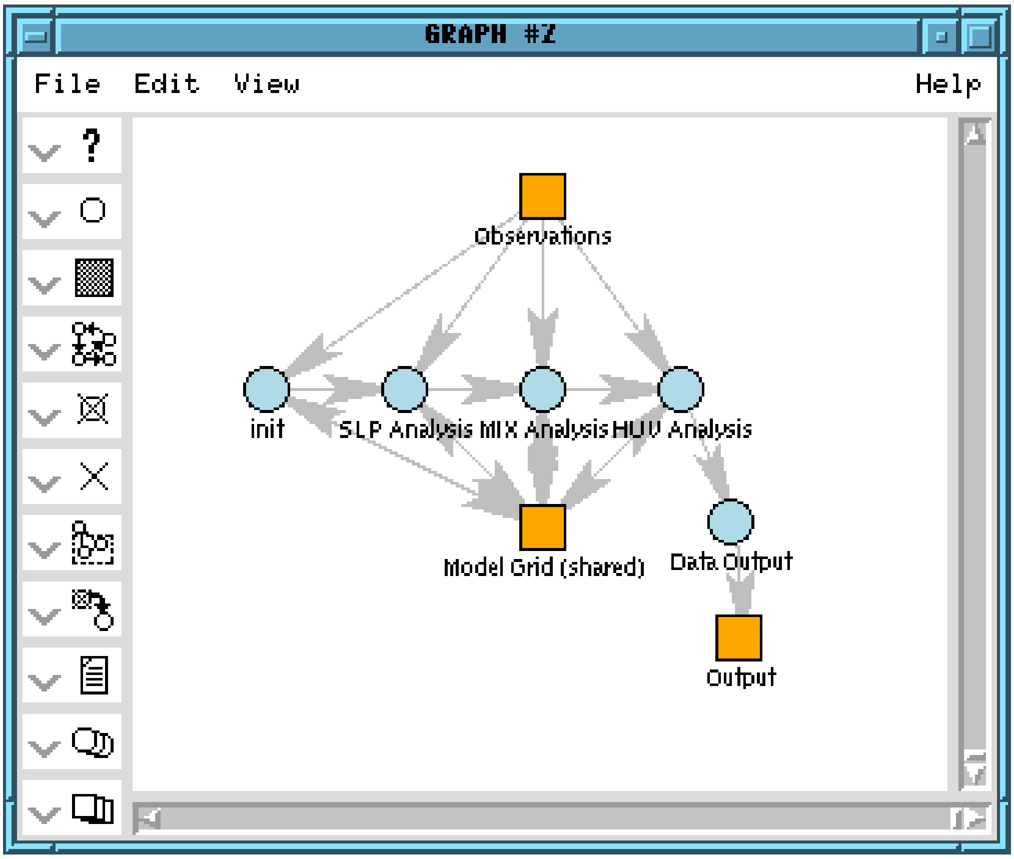

Figure 6: *The window shows the building blocks used in the global program structure*

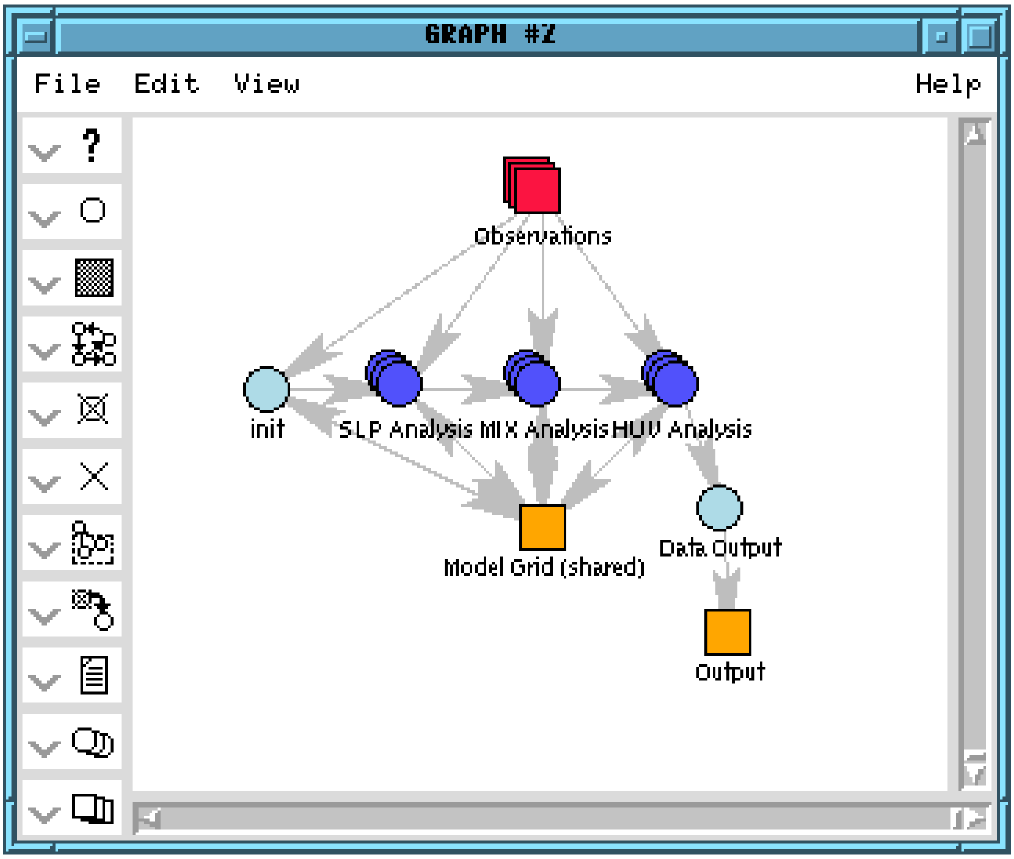

Figure 7: *The window shows how the program is represented after the parallel program blocks have been introduced.*

Figure 6 shows the logical devision of the program. Data is generally shown in rectangles while processes working with on data are displayed in circles. Dependencies between data and processes are displayed with the help of bidirectional edges. For better visualization colors are used in addition to the obvious form distinction.

Once the processes with concurrent nature are defined, they are introduced into the process graph as shown in Figure 7. A parallel process is visualized with multiple circles while data distributed in exclusive way is visualized with multiple rectangles. An example for such distributed data is the block distribution as known from HPF.

After the processes are defined, they have to be mapped onto a real computer to be executed. Restrictions during the code development, .e.g. the code can only be compiled on one machine and is not portable, may limit the number of choices for the mapping. To minimize the overall wallclock time of the program execution dynamic load balancing can be used to map the problem on the different processors and/or computers based on their current load.

To support this strategy, a process monitor keeps track of the status and usage of the machines. Figure 9 shows an example of some system variables[2] monitored to support the mapping strategy.

The load monitor helps also to display performance bottlenecks of the parallel program during its execution on the hardware, while collecting a time space diagram.

In the example depicted, all processes are mapped to an SP2 and a SPARC workstation. In case the processes are written in a portable way, mapping onto other machines is possible too. This way one can execute a parallel program in a heterogeneous supercomputing environment.

[2] here CPU Load, Load Average, and Swap Load

The implementation of the parallel software environment is done in TCL/TK, C++ and Perl. Because of the complexity of the program we convert the program currently to Java. This has also the advantage that a virtual supercomputer programming environment can be integrated in an environment accessible via Internet. Figure 10 shows the hierarchy of the different parts of the software environment. The first layer is the communications library, which provides an interface to MPI. On top of MPI a *Parallel Vector Library* provides parallel vector routines necessary for the parallel OI or similar codes. In addition it includes some parallel sorting algorithms.

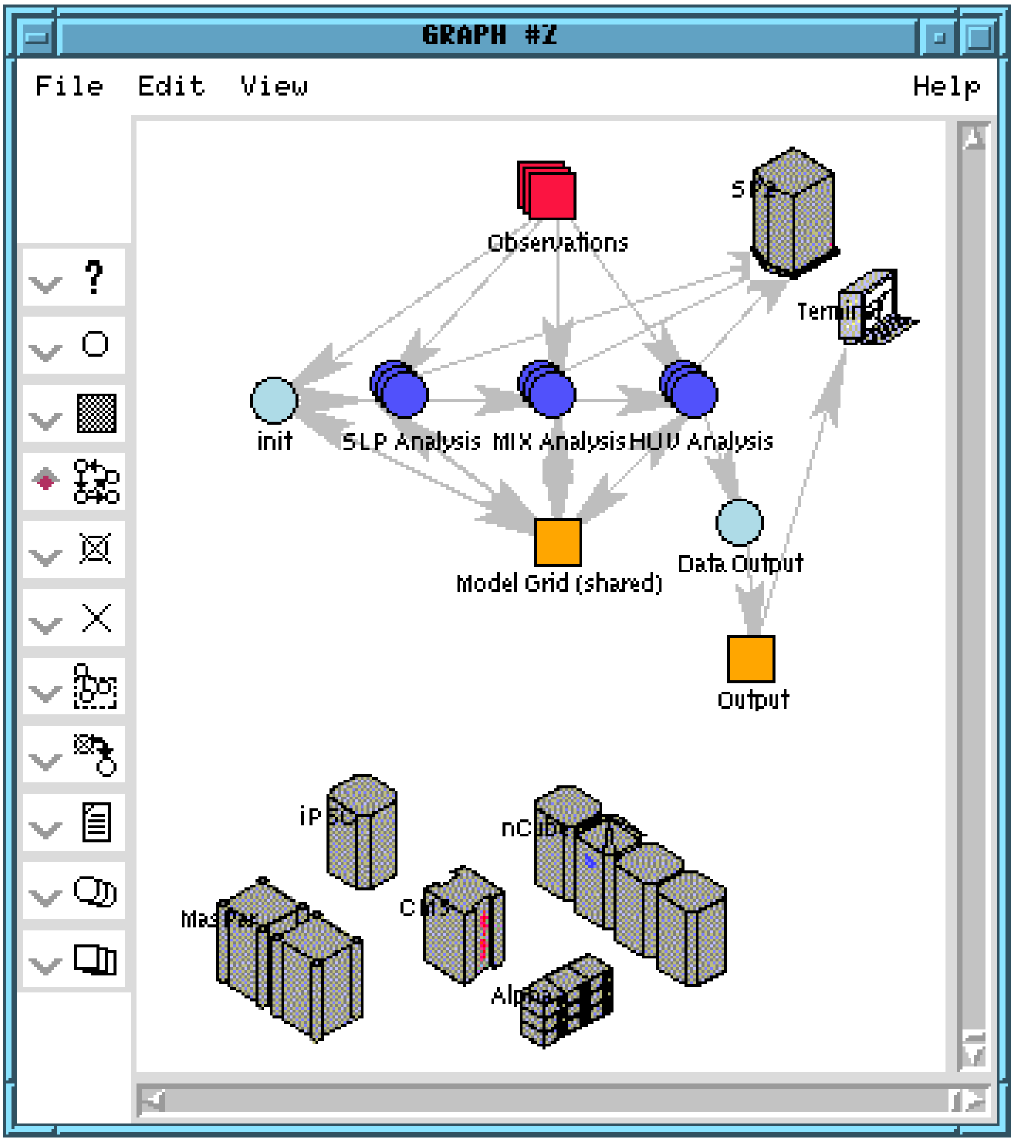

Figure 8: *The window shows the selection of the machines participating in the execution of the program*

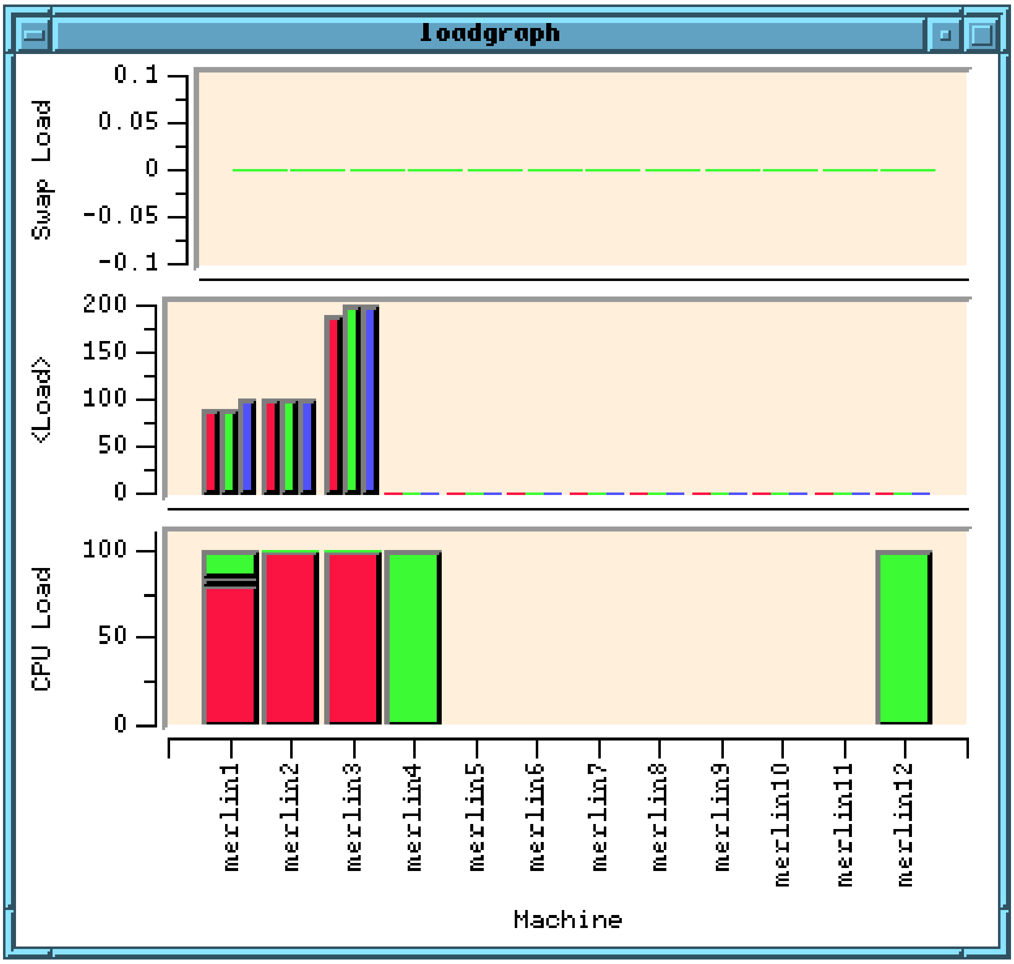

Figure 9: *The window shows the load meter to control dynamic load balancing while executing the code*

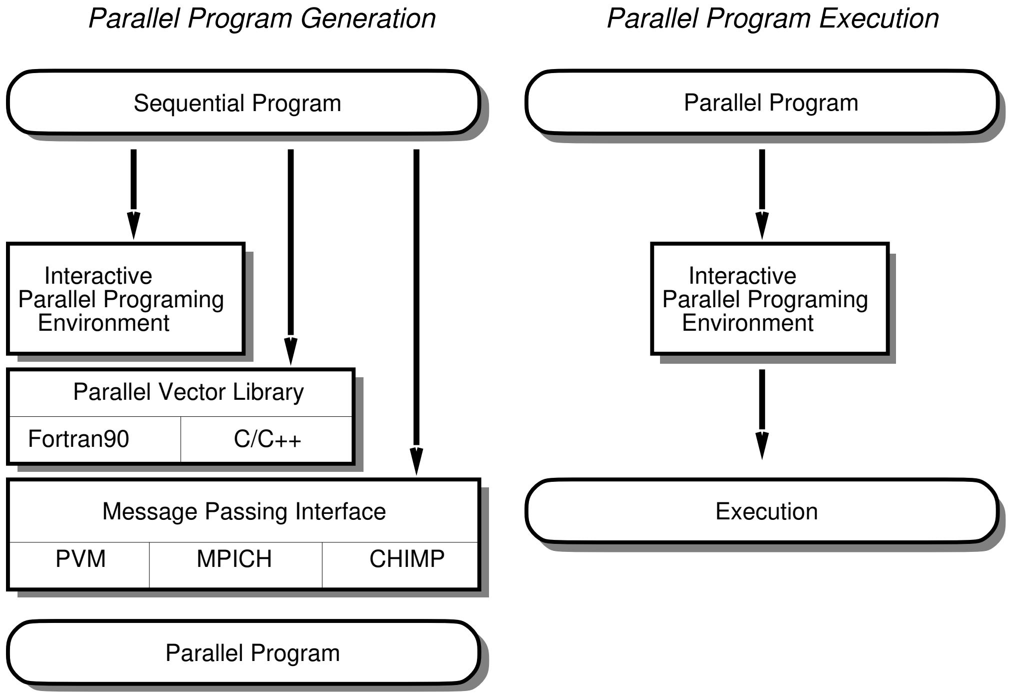

Figure 10: *Hierarchical overview of the parallel programming environment and the relationship to the sequential and parallel program*

4 The Parallel Vector Library

In this section the a parallel vector library is described as used in part of the parallel optimal interpolation algorithm. Its functionality is similar to block distributed vectors as known in High Performance Fortran and Fortran D. To assure maximal portability on all significant MIMD supercomputers the underlying message passing library is MPI.
Elaborate Fortran 90 interfaces exist in order to eliminate the usage of different names for procedures and functions having elementary data types, like *integer real, double precession, and logical*, as parameters. An experimental parallel IO library is in development but is currently unsupported.
The library enables to execute simple vector constructs concurrently on a parallel computer. Similar concepts exists for SIMD and MIMD machines with the help of High Performance Fortran. Unfortunately, the compilers are not yet stable enough to just use HPF. Therefore, we developed all necessary parallel constructs for this particular application in the hope that more stable compilers are available in the next years.
The requirements towards the design of the vector library are

- a simple interface,
- an easy extension of the library, and
- maximum portability on as many supercomputers as possible.

To ensure the requirements the runtime library for parallel vectors is written entirely in FORTRAN 90 and uses MPI as underlying message passing library. The portability is guaranteed while using MPI and the ease of use is guaranteed while providing the extensive set of interface routines to the core algorithms and different data types.
It is intended to extend the library on an as need basis. This includes also the introduction of irregular distributed arrays. Irregular distributed arrays are arrays which are not distributed equally between the processors. allowing to make use of heterogeneous computing platforms. Overall, the library simplifies

- the design of parallel programs,
- the switch to use High Performance Fortran
- the substitution of the basic constructs with custom designed once (e.g. regular vs. irregular arrays).

```
1 real, parallel array, regular :: a
2 real, total, mean
3 PA_ALLOCATE a(30)
4 a = 10
5 (total,mean) = PA_MEAN(a)
6 write (6,*) mean, total
7 PA_DEALLOCATE a
```

Figure 11: *Example program to demonstrate the usage of the parallel vector library*

A simple example shown in Figure 11, illustrates how to use some elementary constructs of the library. The program calculates the mean value of a parallel array. It should be noted that the code does not contain a single message passing routine, making the functionality of the program more transparent.
In the first line a parallel array is declared. The size of the array can be set dynamically during runtime. In the example, the allocation of the array of total size 30 is done in line 3. This 30 memory locations are distributed evenly over the number of processors. In the line 4 a simple assignment is used to indicate that all elements of the array are initialized with the value 10.
In the line 5 two values are assigned simultaneously while executing the function calculating the mean value. In order to calculate the mean value the sum of all the array elements is produced as a side effect, because

DECLARATION of a parallel array of type real, integer, or logical.

- real , parallel array, regular :: a
- integer, parallel array, regular :: a
- logical, parallel array, regular :: a

ALLOCATION of a parallel array of size n

- PA_ALLOCATE a(n)

DEALLOCATION of a parallel array

- PA_DEALLOCATE a

REDUCTION operators of a parallel array. E.g. sum, mean, and standard deviation of its elements.

- sum = PA_SUM(a)
- (sum,mean) = PA_MEAN(a)
- (sum,mean,std) = PA_STD(a)

VECTOR operations are implicit due to Fortran90 semantics. E.g. Assignment, Addition, Subtraction, Multiplication

- c=10, c=a+b, c=a-b, c=a*b

SORTING, RANKING, and REORDERING sorts an array and returns an index rank. Reorders are possible with the ranking returned by the sort routine.

- real, parallel array, regular :: values,
  sorted, reordered, unsorted
- integer, parallel array, regular :: index
- (sorted,index) = PA_SORT(values)
- reordered = unsorted[index]

Figure 12: A summary of some simple commands available in the parallel vector library.

they are often needed together. In *total* the total sum of all the values in the array *a* will be stored, while the mean value will be stored in the variable *mean*. Both variables are declared at the beginning of the program and are of data type real. Once the calculation is performed it is desirable to eliminate the space occupied by the parallel array to free it for other usage. Therefore, a deallocation statement is used in the last line to free the memory space allocated with the parallel array.

Command Summary

An overview of some of the parallel vector routines is given in Figure 12. Information about the type , the usage, and the semantics of the operations is displayed. To simplify some the notation of local loops, the FORALL construct is supported(Figure 13). In order to apply a FORALL loop on a data structure dependent on a parallel array, the PA_FORALL construct can be used.
The Fortran code using the parallel vector library is translated by a source to source code compiler. The compiler can produce sequential code, HPF code, or, as used for our experiments, code using Fortran 90 with message passing extensions. This leads to an easy way of defining parallel programs using vectors, as used by the optimal interpolation algorithm where observations are distributed over several processors.

Code Reduction and maintenance

There is only little difference between the vector library of one elementary data type to another. Therefore, the code for the data types is automatically created with the help of templates similar to C++. The internal representation of the templates is done in m4, a familiar macro processor, and nuweb, a literate programming preprocessor. The templates used for different data types are expanded at compile time and providing a library which is only 1/4th the size, which would be necessary if no templates are used. This reduces the cost of maintenance drastically. Due to the usefulness this concept it would be useful to expand HPF with templates.

5 Results

Results presented here are split in two parts, because the operational optimal interpolation algorithm has been updated recently from version 1.2 to the version 2.0.
First, results for the parallelization of version 1.2s are shown. Findings obtained during the parallelization of the version 1.2s have a direct impact on the parallelization strategy used in the version 2.0mv of the OI code, reducing the software engineering effort drastically.
Both versions are applied on a problem instance defined by a 4 by 5 degree latitude-longitude grid with 14 height levels, resulting in a domain of $144 \times 75 \times 14$ grid points. About 15000 mini-volumes divide the grid points in small subregions. The total amount of observations is approximately 90000.
Most of the experiments are conducted on a SP2 with up to 40 nodes. The computational nodes are based on the IBM RISC System/6000 chip set. The actual model is a 390 with 66.7 MHz clock, also known as *thin node* (66). The processor is also referred to as *Power 2*. The internal characteristics are a 64 KByte data cache, a 32 KByte instruction cache, and a microchannel bus performing at 80 Mbytes/sec. The main memory contains currently 256 Mbytes. An external disk of 1.0GB is available for each node.
Initially, it was important to demonstrate that a parallelization of the algorithm is at all possible. Due to the programming style used in the original algorithm a considerable amount of time was spend to port the sequential code to other platforms. Currently, the code conforms Fortran 90 ANSI standard and will run on a Cray Y/MP, a Cray C90, DEC Alpha Workstations, and IBM RISC Systems 6000. The later one are of upmost importance because state of the art systems with up to 255 nodes will be accessible for the project.

Version 1.2

Because of the complexity of the original code and the limited resources available to parallelize and modify the sequential code, it was clear that a simple approach for the parallelization had to be chosen, instead of a more time consuming reengeneering approach. The sequential parts of the program should be kept unchanged as much as possible, while adding message passing routines at the appropriate places to parallelize the program.

FORALL loops for local arrays

```
real a(n,m,l)
FORALL (i=1:n,j=i:m,l=1:l)
  a(i,j,l) = calculation
END FORALL
```

PA_FORALL loops for parallel arrays

```
real   , parallel array, regular :: a
FORALL (i=1:n)
  a(i) = calculation(i)
END FORALL
```

Figure 13: Additional Syntactical simplifications

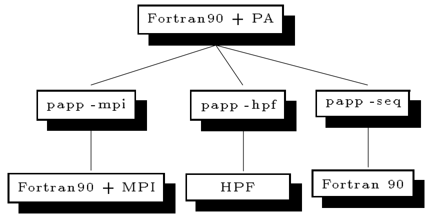

Figure 14: The relationship between the parallel array library and different parallel programming paradigm/languages.

Analysis of the Sequential Program

An analysis of the sequential program was used to indicate hot spots in the algorithm. Parallelizing the program at its hot spots lead to a good overall performance. The computational intense parts of the program are the HUV quality control with about 40% and the HUV matrix solve with about 48% of the overall computation time (See Figure 26).
The sequential analysis of the original program reviled many opportunities to speedup the program. While optimizing the sequential program we could improve the overall performance by about 40%, resulting in 2400 seconds for the overall runtime of the sequential algorithm. Performance improvements are mainly based on modifications of the quality control algorithm. Here a new algorithm to collect the observations in the region of influence was used. The algorithm is based on a range search algorithm. First, the observations are sorted in latitude direction. Then, the search for observations can be aborted once a particular observation with a latitude distance, defined by the search radius, is reached. Therefore, the search routine does not check all observations but only those placed in a 20 degree latitude band around he original observation (based on the 1600km radius).

The Parallel Program

To parallelize the given algorithm efficiently, a data parallel and task parallel approach is used in conjunction. Data objects essential for the parallelization are

- the model data specified by the longitude-latitude grid,
- the observations,
- and the mini-volumes dividing the grid in small subregions.

A careful analysis of the optimal interpolation algorithm reveals that mini-volumes and observations can be used to develop a task parallel algorithm. On the matrix solve level, mini-volume divide the mathematical core calculation, naturally in small independent functional units.
In contrast to the matrix solve, the main iteration of the quality control takes place over the observations. They build the foundation for a natural functional division of the quality control.
The parallel formulation of the algorithm using a task parallel approach is therefore quite simple and introduced in the high level algorithmic description in Section 1.
The conditions $\in$ *this processor* in line 4 of the gross-check, in line 2 of the buddy check, and 2 of the matrix solve control the parallel execution of the computational block in a particular processor.
Unfortunately, observations from other processors and are necessary to perform the calculation specified in the parallel loop bodies. The data is irregular of nature, thus complicating the parallelization.
Under the assumption that the data used in the bodies can be predetermined, it is best to prefetch them in order to avoid expensive communication routines during the execution of the loop.
Dependent on the data distribution different strategies for prefetching the data exists. For the version 1.2s a striped and a data balanced distribution are used. Then, prefetching is based on obtaining all observations in a small border stripe of the neighboring processors.

Striped Decomposition

First, the *stripped* data distribution is considered. In HPF this distribution is also known as block distribution. Here the term striped decomposition is preferred because the block distribution is done in one dimension, and each processor is assigned an equal sized latitude band (stripe).
In contrast to the simple block distribution, known from HPF, a ghostpoint area must be defined to incorporate data from neighboring processors. Introducing, perfetching of data contained in a ghostpoint region has the advantage to avoid costly communication during the calculation. The size of the overlap region is determined by the radius of the mini-volumes. In the current operational system the radius is 1600km leading to an overlap region of 10 degrees in latitude direction. Figure 15 shows the two cases which can occur for a mini-volume:

1. All necessary observations are embedded in the assigned processor.
2. Some necessary observations are embedded in neighboring processors.

One limitation of this parallel OI algorithm is that the ratio between overlap region and actual computational region should be small to provide a high degree of parallelism. This is the case for small numbers of processors.

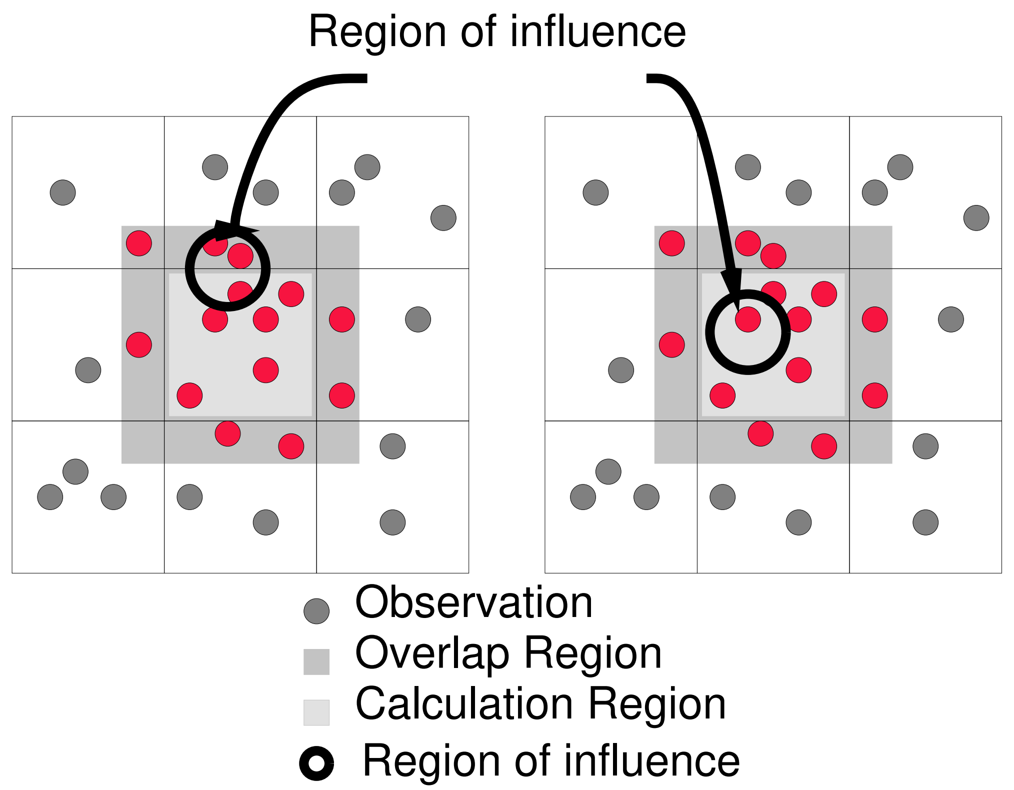

Figure 15: *Overlap region between processors. The size of the region is determined by the radius of influence, e.g. 1600km*

The time used to perform the quality control is strongly correlated to the number of observations available in the region assigned to a processor. This motivates a task and data decomposition based on a bisectioning algorithm[18], in order to obtain a better performance. Because this strategy does not necessarily guarantee load balance the term *data balanced* strategy is used. Nevertheless, this strategy is in the class of a dynamical load balancing strategies, determining the exact decomposition during runtime. The *data balanced* strategy distributes on each processor an approximately equal number of observations.

While the strategy is expected to perform very good for the quality control, it performs less efficient for the matrix solve. The reason being, that the times performing the matrix solves are not only strongly to the number of observations in the region assigned to a processor but also the number of mini-volumes in the particular region.
A comparison between the striped and data balanced decomposition is shown in Figures 16-21 for 10,20 and 40 processors. It is obvious that fewer observations are located, for example, on the poles in contrast to, for example, Europe. Therefore, the data balanced stripes on the poles are much larger than in Europe. This leads consequently to the problem that different numbers of mini-volumes are assigned over different latitude bands on the globe, thus, introducing load imbalance for the matrix solve.
The Figures 22-25 display the results for the most computational intense parts of the optimal interpolation algorithm. They are the HUV quality control abbreviated with *delhuv* and the ZUV-analysis abbreviated with *zuvanl*. For more algorithmic details it is referred to the source code [13]. Figures 22 and f:huv-perf-load show the timings and the speedup obtained for both distributions with different numbers of processors.
Performance data for the stripped decomposition is displayed in Figure 22. An important result from the experiments is that the time used for the quality control and the matrix solve is very close to each other. This influenced the time spend to parallelize the OI code equally between the quality control and the matrix solve.
For small numbers of processors ($<= 10$) it was possible to achieve a very good speedup for both algorithms. To achieve a very good performance for small numbers of processors is especially of interest, since one of the computers considered for operational use, is based on several Cray computers connected via a HIPPI network.
For the data balanced decomposition a super linear speedup cold be achieved during the quality control. This is motivated by the following facts. First, the fraction between observations in ghostpoint areas and the other observations stored in the processor is large. Second, to determine the neighbors influencing an observation a modified search routine is used. The performance gain while using the improved search algorithm is the biggest the more data points are stored in a processor. This is the case for large latitude bands, as obtained by using few processors. Third, while increasing number of processors the search of observations in the ghostpoint area becomes dominant and the speedup slows down. Forth, regions which contain a only a few observations perform the matrix solves very quickly. This is not the case for the striped decomposition, but it is better for the data balanced distribution. While using 40 processors the performance of the HUV matrix solve is almost the same for both decompositions (Table 2).
The performance can be improved further, while introducing a block distribution[20]. Instead of using the block distribution, a cyclic distribution is used for the version 2.0mv, simplifying the parallelization effort drastically while increasing the memory consumption.

5.1 Version 2.0

The switch to the new version was necessary to improve the performance of the OI algorithm drastically. Besides better performance, it is more stable, and produces better scientifically sound results. The performance increment is obtained while using LAPACK routines for the matrix solves, and linking the program with a machine optimized versions of BLAS. Another important change was to eliminate and rewrite certain diagnostic functions in the quality control. These routines inherently sequential in nature and contain many file input and output operations, which were difficult to parallelize.
As depicted in Figure 26, the new OI algorithm has different hot spots in contrast to the old OI algorithm. In the old version the times for the huv quality control and matrix solve were almost the same. In the new version the zuvanl constitutes of 83% of the total runtime of the algorithm, while the huv quality control uses only 5%. The total runtime could be improved by a factor of 4.
Table 1 shows the runtime of the sequential algorithm compiled with different options on SP2 nodes and DEC Alpha workstations. Unfortunately, a valid license for the optimized BLAS and LAPACK libraries was not available on the DEC Alpha workstations, the performance on this machine has to be viewed as preliminary. More interesting are the times for the SP2 because of the availability of larger machines. Unfortunately, it is not possible to compile the OI code on an RS6000 with an optimization level higher than two without influencing the numerical stability of the computation. Inlining, performed with the option -pipa improves the

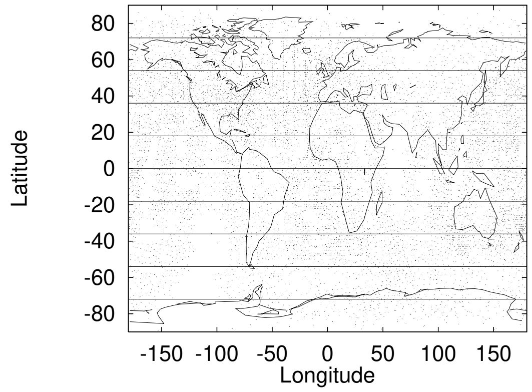

Figure 16: *Stripped data decomposition onto 10 processors*

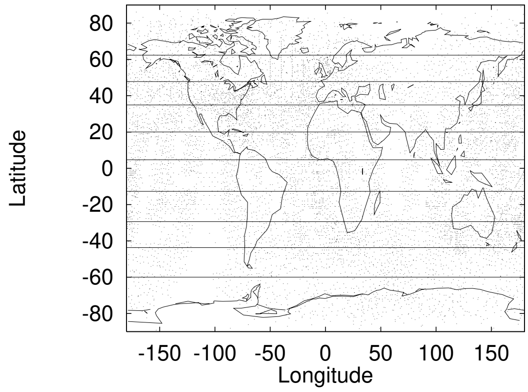

Figure 17: *Data balanced stripped decomposition onto 10 processors*

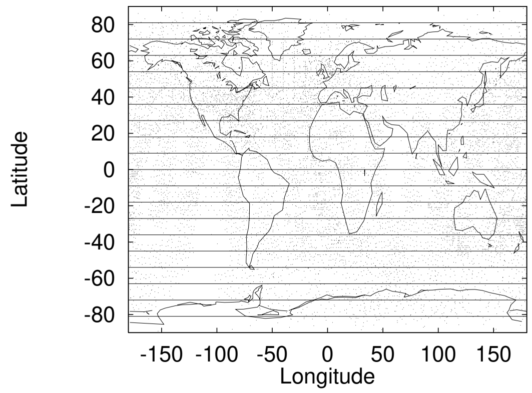

Figure 18: *Stripped data decomposition onto 20 processors*

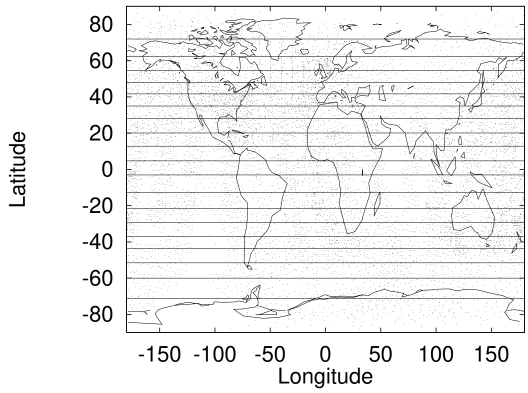

Figure 19: *Data balanced stripped decomposition onto 20 processors*

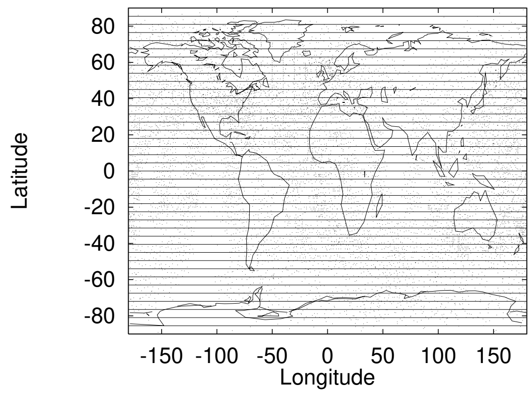

Figure 20: *Stripped data decomposition onto 40 processors*

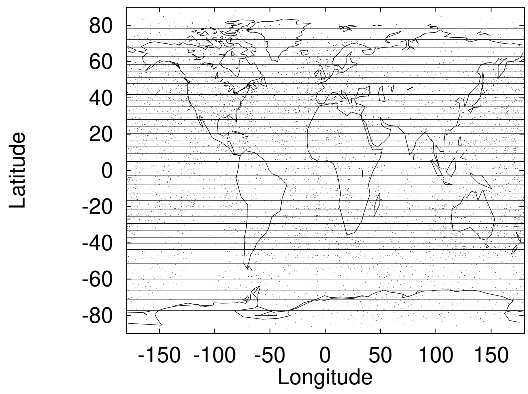

Figure 21: *Data balanced stripped decomposition onto 40 processors*

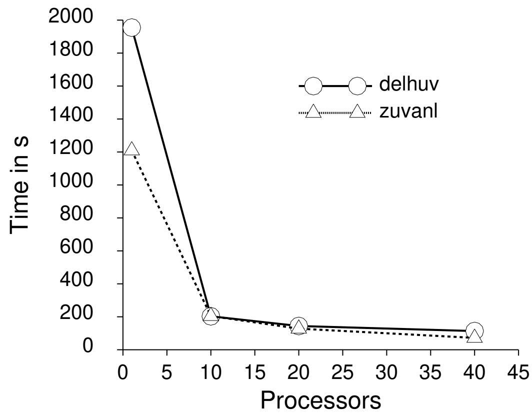

Figure 22: *The calculation time vs. the number of processors used for the HUV analysis, using the striped decomposition*

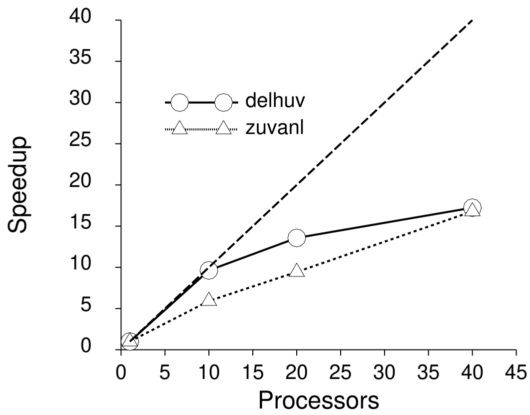

Figure 23: *The speedup vs. the number of processors used for the HUV analysis, using the stripped decomposition*

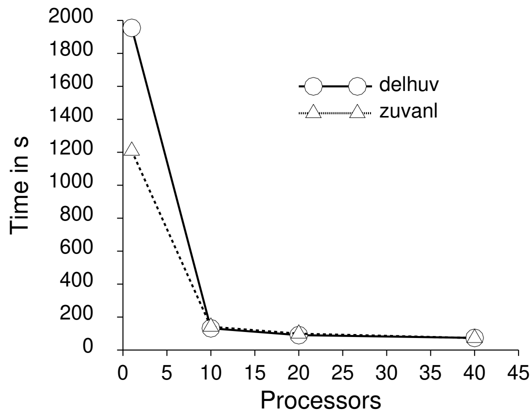

Figure 24: *The calculation time vs. the number of processors used for the HUV analysis, using the data balanced decomposition*

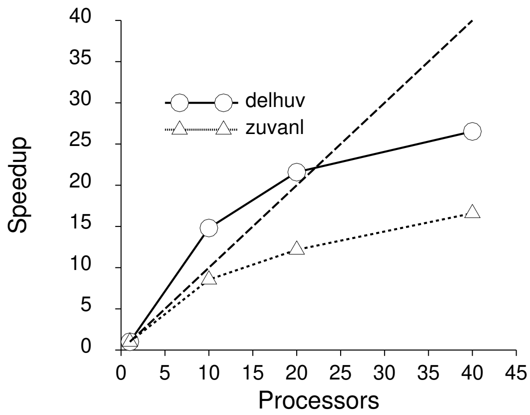

Figure 25: *The speedup vs. the number of processors used for the HUV analysis, using the data balanced decomposition*

runtime about 23%. Since this optimization step takes a long time during the compilation the timings given for the parallel code are obtained only with the optimization level two on thin nodes of an SP2. In case wide nodes should be available, the performance can be improved by another 22%. This increased performance gain does not influence the parallelization strategy of the OI code of version 2.0.
An overall comparison of the speedup between the different versions using different decompositions is given in Table 3.
As mentioned before, the load imbalance problem is differently solved in version 2.0mv in contrast to the older version. Instead of decomposing the domain in stripes or applying the data balanced decomposition, a cyclic decomposition is implemented. The cyclic decomposition preserves the data balance in a natural way. Figure 27 verifies that the cyclic decomposition solves the problem of the load imbalance, comparing the runtimes for the quality control using the striped and the cyclic decomposition. The following preconditions are responsible for a better load balance using the cyclic decomposition:
First, the time to calculate the quality control for one observation depends on the number observations which are located in the region of influence (number of neighbors).
Second, from the distribution of the observations it is clear that the number of neighbors varies over the domain. While there are clusters which have many observations and therefore many neighbors, other regions do not.

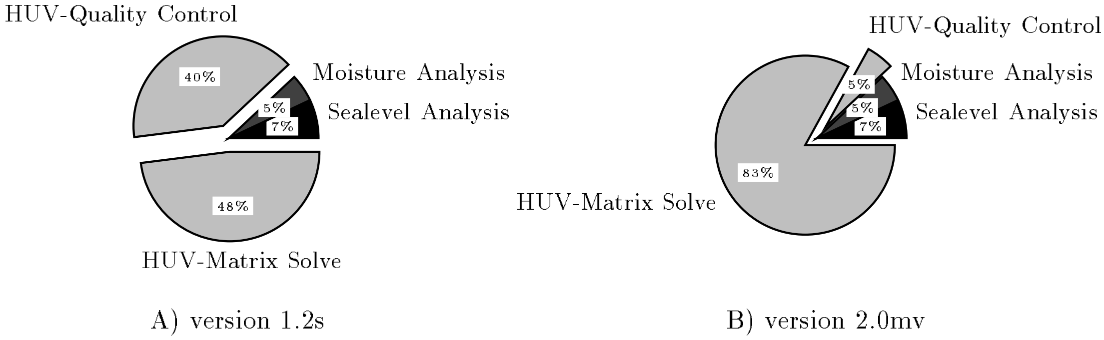

Figure 26: Fractions of the computational intense parts of the sequential optimal interpolation algorithm for the versions 1.2s and 2.0mv

Table 1: *Performance of the Sequential Program*

	SP2	SP2	SP2	DEC Alpha	DEC Alpha[1,2]
node type	thin	thin	wide		
comp. options	-O2 -pipa	-O2	-O2	-O2	-O4
data IO	1.21	1.91	1.22	6.55	6.53
delhuv	44.38	41.51	38.72	82.26	78.80
zuvanl	621.39	818.23	639.63	660.55	590.55
total zuv time	667.05	861.80	679.64	749.73	675.93

Note 0: all times are given in s
Note 1: -allign commons, no performance improvement.
Note 2: -O4 -math math_library fast, not much improvement

Third, the number of grid volumes should be approximately equal to avoid extensive load imbalance in the matrix solve.
For a striped decomposition, the condition to maintain approximately an overall equal number of neighbors in the different processors, is violated leading to load imbalance.
For the data balanced decomposition the third condition is violated. One cold choose a data balanced decomposition for the quality control and a stripped decomposition for the matrix solve, but this would still violate the second condition.
An effective alternative to a region based decomposition (e.g. the striped and data balanced decompositions) is the cyclic decomposition. For the quality control the observations are assigned in cyclic fashion in each processor, similar to a cyclic decomposition as known from HPF. Analog, mini-volumes are assigned in cyclic fashion to different processors for the matrix solve loop.
The cyclic decomposition has the advantage that conditions one to three are fulfilled, resulting in a simple load balance algorithm for the quality control. The only problem remaining, is to show that the execution

Table 2: *Comparison of the runtime of the versions 1.2 and 2.0 using 40 processors using different decompositions.*

	version 1.2	version 1.2	version 2.0
decomposition	striped	data balanced	cyclic
delhuv	87.30 s	56.30	6.46 s
zuvanl	55.47 s	55.44	25.65 s

Table 3: *Speedup of the rspeedup of the versions 1.2 and 2.0 using 40 processors using different decompositions.*

	version 1.2	version 1.2	version 2.0
decomposition	striped	data balanced	cyclic
delhuv	1.0	1.6	13.5
zuvanl	1.0	1̃.0	2.2

times for all mini-volumes assigned with the cyclic decomposition to the different processors is almost the same. This is unfortunately not the case, as shown later, justifying the use of more intelligent dynamical load balancing strategy.

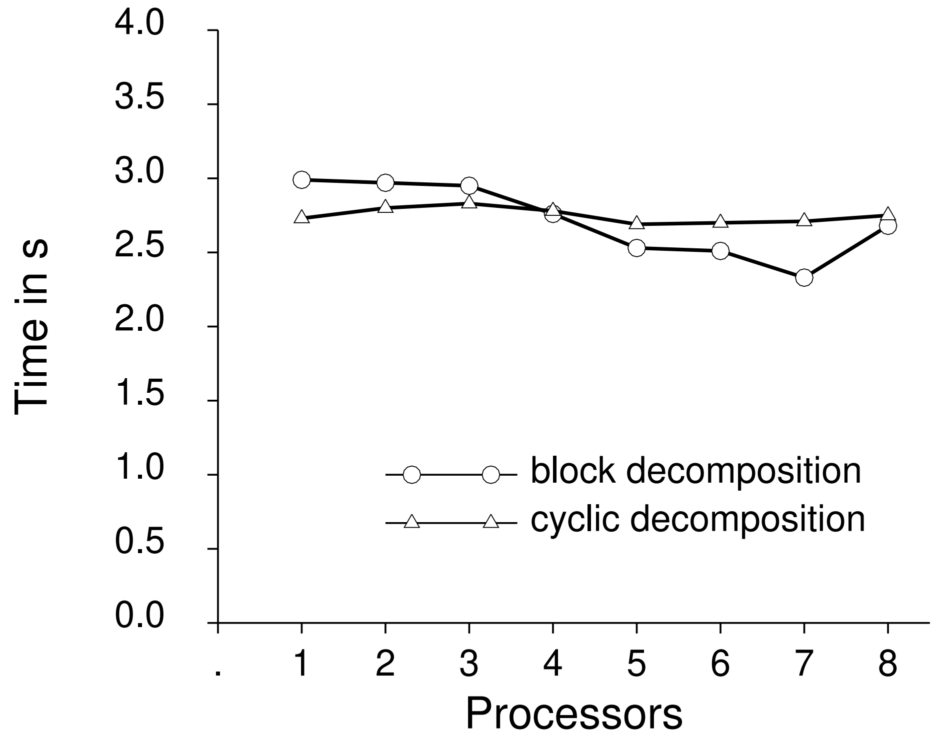

Figure 27: *Load imbalance using 8 processors for the quality control check using the block and cyclic decomposition*

The actual number of observations, which are reaccepted during the quality control, is rather small (as shown in Figures 28 and 29). Thus, there is almost no overhead involved for redistributing reaccepted observations to other processors, after the quality control is completed. The additional redistribution step is necessary to update observations which are members of ghostpoint areas in other processors.

Due to the parallelization of the a problem in the sequential quality control program sequential program was reveled. In the main loop of the buddy and gross check a reaccepted observation is immediately added to the set of valid observations therefore influencing the next iteration of the loop over all observations.

This strategy has the disadvantage that the quality control is dependent on the order of the observations and a parallelization would not be possible. To eliminate the data dependencies between the loops the new strategy reaccepts observations only at the end of the quality control. Both strategies have scientific advantages and disadvantages, which should be further explored. This resulted in plans at the DAO to redesign the current quality control algorithm.

Naturally, the quality control is only scalable for processor numbers up to the number of observations which are suspect to the quality control check. In the example used about 300 observations had to be checked, and about 30 of them are reaccepted(Figures 28 and 29). If the number of quality control failures are not significantly larger than the number of processors the performance will degrade.

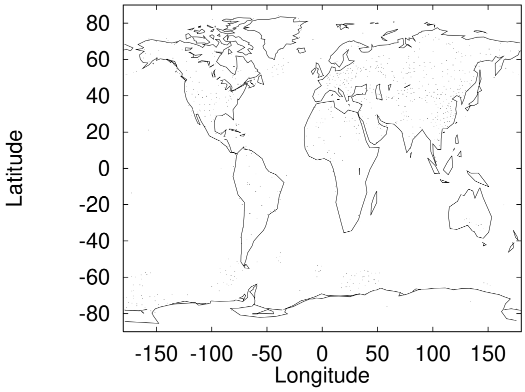

Figure 28: *Observations which have failed the HUV quality control*

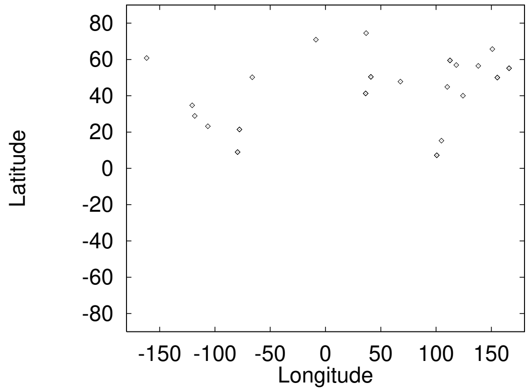

Figure 29: *Observations which failed initially the HUV quality control, but were reaccepted*

Matrix Solve Figure 30 and 31 display the runtimes on different processors for the quality control (delhuv) and the matrix solves (zuvanl) and their ranges (compare also Table 5). Thus, proving that the cyclic decomposition is ideal for the quality control. Only two processor in the matrix solve algorithm have significantly different load in contrast to the others. These processors contain the polar regions which will be differently handled than the rest of the globe. Assigning fewer mini-volumes to the processors containing the polar regions solves the problem of load imbalance. A scalable parallel algorithm for the optimal interpolation algorithm is the result.

Table 4: *Performance on 8 SP2 Processors*

comp. options		-O2		
	Efficiency	max	min	median
data IO	0.92	1.30	1.21	1.25
delhuv	0.61	8.48	8.28	8.35
total zuvanl	1.13	95.55	86.21	90.25

Note 0: all times given in s
Note 1: communication done over Ethernet

Table 5: *Performance on 40 SP2 Processors*

	delhuv	total zuvanl
mean	6.18	17.17
minimum	5.78	15.74
maximum	6.46	25.65
variance	0.02	3.87
σ	0.12	1.97

Note 0: all times given in s
Note 1: communication done over Ethernet

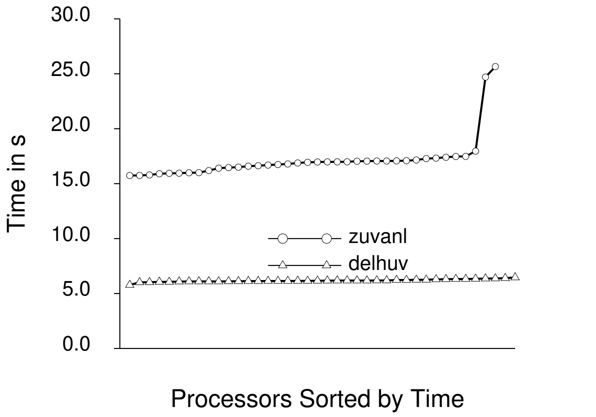

Figure 30: *Forty processors and load imbalance of the cyclic decomposition*

Figure 31: *Forty processors and the range of the load Imbalance of the cyclic decomposition*

References

[1] Wayman E. Baker, Stephen C. Bloom, John S. Woollen, Mark S. Nestler, and Eugenia Brin. Experiments with a Three-Dimensional Statistical Objective Analysis Scheme Using FGGE Data. *Monthly Weather Review*, 115(1), 1987.

[2] V. Bjerknes. *Dynamic Meteorology and Hydrography.* Carnegie Institute, Gibson Bros., New York, 1911.

[3] Stephen E. Cohn, N.N. Sivakumarun, and Ricardo Todling. Experiments with a Three-Dimensional Statistical Objective Analysis Scheme Using FGGE Data. *Monthly Weather Review*, pages 2838–2867, Dec. 1994.

[4] Roger Daley. *Atmospheric Data Analysis.* Cambridge Atmospheric and Space Science Series, Cambridge University Press, 1991.

[5] A. Eliassen. Provisional Report on Calculation of Spatial Covariance and Autocorrelation of the Pressure Field. Technical report, Videnskaps-Akademiet, Institut for Vaer og Klimaforskning, Oslo, 1954.

[6] L. Gandin. *Objective Analysis of meteorological fields.* Gridoment, Leningrad, 1963. translation to English: Israel Program for Scientific Translation, 1965.

[7] Gregor von Laszewski. Interactive Parallel Program Generation. Number SCCS , June 1996.

[8] Jing Guo and Arlindo da Silva. Computational Aspects of Goddard's Physical-Space Statistical Analysis System (PSAS). In *Second UNAM-CRAY Supercomputing Conference on Numerical Simulations in Environmental and Earth Sciences.* Mexico City, Mexico, June 1995. updated in Nov. 1995.

[9] P. M. Lyster, S. E. Cohn, R. Menard, L.-P. Chang, S.-J. Lin, and R. Olsen. An Implementation of a Two Dimensional Kalman Filter for Atmospheric Chemical Consituent Assimilation on Massivley Parallel Computers. *submitted to: Monthly Weather Review*, June 1995. NASA GSFC Data Assimilation Office, Greenbelt, Maryland.

[10] James Pfaendtner, Stephen Bloom, David Lamich, Michael Seablom, Meta Sienkiewicz, James Stobie, and Arlindo da Silva. Documentation of the Goddard Earth Observing System (GEOS), Data Assimilation System - Version 1. NASA Technical Memorandum 104606, Vol.4, NASA GSFC Data Assimilation Office, Greenbelt, Maryland, Jan. 1995. (ftp).

[11] J.W. Pfaendtner, R. Rood, S. Schubert, S. Bloom, D. Lamich, M. Seablom, and M. Sienkiewicz. The Goddard Global Data Assimilation System: Description and Evaluation. *Submitted to Mon. Wea. Review*, 1993.

[12] L. Richardson. *Weather Prediction by Numerical Process.* Cambridge University Press, 1922.

[13] Contact: {rood,lamich}dao.gsfc.nasa.gov. Source Code of the Assimiation System based on Optimal Interpoaltion. (Source), 1994,1995. Version 1.2 and Version 2.0mv.

[14] M.S. Seablom. Experiments with new quality control techniques in the NASA Optimum Interpolation Analysis System. In *Preprints, International Symposium on Assimilation of Observations in Meteorology and Oceanography*, volume WMO. Preprint volume, pages 628–630, Clermont-Ferrand, France, 1990.

[15] 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. In *International Symposium on Assimilation of Observations, Tokyo, Japan*, March 1995.

[16] Lawrence L. Takacs, Andrea Molod, and Tina Wang. Documentation of the Goddard Earth Observing System (GEOS), General Circulation Model - Version 1. NASA Technical Memorandum 104606, Vol.1, NASA GSFC Data Assimilation Office, Greenbelt, Maryland, Sep. 1994. (ftp).

[17] Kevin E. Trenbreth, editor. *Climate System Modeling.* Cambridge University Press, 1993.

[18] Gregor von Laszewski. Object Oriented Recursive Bisection on the CM-5. Technical Report SCCS 476, Northeast Parallel Architectures Center at Syracuse University, April 1993.

[19] Gregor von Laszewski. Preliminary Performance of a Parallel Interpolation Algorithm. Technical Report SCCS 713, Northeast Parallel Architectures Center at Syracuse University, December 1995. first edition in June 1995.

[20] Gregor von Laszewski and et al. Design Issues for the Parallelization of an Optimal Interpolation Algorithm. In *4th Workshop on parallel processing in Atmospheric Science, Edinburgh, UK*, Nov. 1994. (ftp)