A Parallel Data Assimilation System and its Implications on a Metacomputing Environment

by

Gregor von Laszewski

B.S, University of Bonn, Germany, 1987
M.S, University of Bonn, Germany, 1990
Diplom Informatiker

Abstract of Dissertation

Submitted in partial fulfillment of the requirements of the degree of
Doctor of Philosophy in Computer and Information Science
in the Graduate School of Syracuse University

October 1996

Abstract

Grand challenge applications are a major motivation to advance research in scientific computing, theoretical computer science, computer architecture, and other fields.
Even though a lot of research has been done while parallelizing climate models, little has been done in the area of atmospheric data analysis.
The first contribution of this thesis is the analysis of a production code for four dimensional data assimilation. Whether the specification of an efficient parallel algorithm is possible, was evaluated. Second, a parallel algorithm has been developed utilizing MIMD distributed message passing machines. Strategies to further optimize the algorithm are given. One of the parallel algorithms specified, can be implemented in a similar manner on different parallel computing architectures, utilizing data parallel and message passing parallel architectures.
A major impact of the dissertation is the definition of a deterministic quality control algorithm, which is an integral part of the assimilation system. This new algorithm will be incorporated in a future production version of the assimilation system.
The other contributions of the dissertation are based on the design of a metacomputing environment simplifying the specification and instantiation of parallel programs on diverse computing platforms. Programming the components of the metacomputer is achieved with the help of a graphical user interface. The interface allows one to design programs in the dataflow concept. In contrast to other approaches, it proposes the use of a dynamical dataflow model instead of a static dataflow model.
The dynamical dataflow concept enables one to facilitate loosely coupled and tightly coupled components of the metacomputer. The differentiation between tightly and loosely coupled metacomputer components guarantees the generation of efficient message passing programs and a simple job distribution facility for jobs to be issued in the metacomputing environment.
Simplicity and intuition has been used for the design of the interface to allow a small learning curve. Due to the generality of the dataflow concept, a program designed with the environment can be translated to different programming paradigms.

A Parallel Data Assimilation System and its Implications on a Metacomputing Environment

by

Gregor von Laszewski

B.S, University of Bonn, Germany, 1987
M.S, University of Bonn, Germany, 1990
Diplom Informatiker

Dissertation

Submitted in partial fulfillment of the requirements of the degree of
Doctor of Philosophy in Computer and Information Science
in the Graduate School of Syracuse University

October 1996

Approved _____

Date _____

Contents

List of Figures

List of Tables

List of Programs

Preface

To my parents.

Diese Arbeit ist meinen Eltern gewidmet. Ich bin Ihnen dankbar für Ihre Unterstuzung, ohne die es mir nicht möglich gewesen wäre nach Amerika zu kommen und, weder diese Arbeit zu beginnen, noch erfolgreich beenden zu können.

Acknowledgment

First, I would like to thank my advisor, Professor Geoffrey C. Fox, for his valuable discussions and advices, as well as, his inspiring lectures at the Computer Science Department. I am grateful, that he gave me the opportunity to explore, in depth, different fields in computer science, as well as, computational science.

The dissertation would not have been possible without the cooperation of several researchers at Northeast Parallel Architectures Center(NPAC) and at Goddard Space Flight Center(GSFC) in Greenbelt, MD.

I am grateful to Miloje Makivic for his support and valuable comments during the duration of the NASA Four Dimensional Data Assimilation Project.

In the Data Assimilation Office at GSFC, I am grateful to Peter Lyster for his guidance, the many helpful comments and fruitful discussions, as well as, his support in many other aspects. I would like to thank Mike Seablom, who provided me with version 1.2s of the sequential optimal interpolation algorithm and shared with me his office at GSFC for over half a year. He helped me get acquainted with the many undocumented lines of the source code. I am grateful to David Lamich for providing the new version 2.0mv of the OI code and a consistent input data set, and James Stobie for his efforts in improving the software quality control at DAO. Both were valuable sources of information. I would like to thank all others which helped to improve my understanding of the different algorithms available at DAO, e.g. Arlindo DaSilva, Ying Guo, and Steve Cohen.

I am especially grateful to Richard B. Rood for his support and hospitality during several visits at NASA Goddard Space Flight Center as part of the Universities Space Research Association (USRA).

At NPAC, I am thankful to the systems staff for providing me with the necessary demanding computational resources and the enormous amount of disk space to store the input data sets. Facilities from the Northeast Parallel Architectures Center, Cornell Theory Center, Maui High Performance Supercomputer Center, Goddard Space Flight Center, and the Jet Proportion Laboratory were used to conduct the experiments.

This project has been funded by NASA High Performance Computing and Communications (HPCC) Earth Science Project.

Chapter 1

Introduction

In the last decade, dramatic advances in software and hardware have changed the landscape for computing. Today, personal computers have reached a performance level which could have only been dreamed of, a couple of years ago. The increase in processor speed is accompanied by an increase in available memory to economical prices. New graphic cards let a PC perform at almost workstation speed, for graphics applications. On the high end, vector-supercomputers are outperformed by distributed memory parallel architectures.

Besides the development in hardware, the changes in the software engineering are marked by the acceptance of object oriented programming concepts and languages in the software design. Even Fortran77 has been superseded by a new standard for which mature compilers were introduced during the last years. Incooperating a module concept, already introduced in modern computer languages decades ago, will simplify defining reusable software components. The introduction of vector constructs is especially of interest for the high performance computing community. It enables the user to design programs suitable for vector and MIMD parallel computers. A further key event, in the last few years, is the standardization of High Performance Fortran(HPF) . New language constructs, supporting the distribution of data in an efficient and transparent way, make it possible to further simplify parallel programming and utilize the new and relatively inexpensive MIMD computers, in a more or less, portable way.

Computer networks have evolved from local area networks, over medium area networks, to Wide area networks. The hardware base has been established for the distribution of inform-

ation in a global "World Wide Web". The World Wide Web (WWW) has established itself as a functioning computing environment, accessible by the ever increasing number of online users. Starting from the desire to provide a framework for exchanging data between scientists, it has reached the potential to become the computing platform of the future. Hardware advances in the network technology, like the introduction of the ATM technology (Asynchronous Transfer Mode), provide the necessary backbone for the information exchange between computers. Today, it is important to reduce the latency between communicating computers further, and to increase the bandwidth of the connections on the Internet, in order to avoid congestion on the net.

Currently, the WWW is most frequently used for exchanging data, and allowing online users to access information stored at remote sites. The well known fact that computers are idle during off peak hours, provides an enormous resource of computational power. Accessing the unused CPU cycles in an efficient way and distributing data on which calculations are performed, is a topic of ongoing research at many institutions. The utilization of the resources of the WWW can be expanded not only to the PC level, but also to the level where several supercomputers build the computational nodes. The facilities managing the resources of many cooperating computers is known as the *metacomputing environment*. The cooperating components of the metacomputing environment are referred to as the *metacomputer*. Without doubt, a functioning metacomputer will influence many different research fields. The more resources are available to the scientific community, the bigger the problems are to be solved or the faster solutions can be achieved.

The development of this new infrastructure has been strongly driven by the scientific research community. One of the problems scientists face is solving grand challenges[35]. Grand challenge problems are problems which have to be solved on parallel computers, if at all. Combining several supercomputers available on the WWW will provide more computational resources for researchers involved with grand challenge applications.

An example of such a problem, is the development of a Four Dimensional Data Assimilation System as used in atmospheric science, as well as, oceanography. It is used to find initial conditions for climate and weather models, and to verify the quality of a climate model, in regards to real observed data during the past decade. The field of data assimilation is as old as the definition of a numerical climate model.

Only recently, sufficient parallel computational resources are available to impact the field of

data analysis and climate modeling. With the increased computational power, more reliable forecasts are possible. In the near future, it is predicted that the data assimilation will gain more and more importance. Thus, the need for faster assimilation systems will follow. This can be achieved by developing an efficient parallel algorithm. Once a parallel program is written and the data is produced, it has to be distributed to other researchers. The WWW is an ideal communication media to do this.

1.1 Research Objectives

The acceptance of massively parallel MIMD machines in the computer science research and industry communities is based on economical feasibility and good performance. It is widely accepted that massively parallel algorithms can outperform existing algorithms executed on vector supercomputers.

Even though a lot of research has been conducted while parallelizing climate models, little has been done in the area of atmospheric data analysis. Besides using parallel programming paradigms unfamiliar to the atmospheric scientist, the new computing environments are too complex to employ them easily. The variety of different systems makes it even more complicated for an atmospheric scientist.

The Dissertation has the following objectives:

First, a very popular data analysis method is analyzed on its feasibility of parallelization. If it is possible to find a parallel algorithm, it should be simple but scalable. Problems connected to the parallelization methods should be outlined, in order to support the parallelization of other assimilation strategies.

Second, a method for simplifying the future development of parallel codes in the field of atmospheric data analysis and other grand challenge projects should be found. The simplification should be based on specifying a parallel algorithm, which can be mapped onto different computing platforms. The handling of the computing platforms should be simplified.

1.2 Organization of the Dissertation

Due to the interdisciplinary research performed, this dissertation consists of two parts. Each part begins with a small introduction followed by the research conducted.

We begin by presenting in Chapter 2 the motivation for the data assimilation in atmospheric science. This helps the reader to get acquainted with the technical vocabulary used in atmospheric science. A simple mathematical model describing the major concepts of the data assimilation algorithm, is transferred into high level code fragments. The code fragments are used in later chapters as basis for the parallelization. Issues related to the quality of the observation data, which are an essential part of the calculation, are explained. The algorithm parallelized is used in the NASA Data Assimilation Office, for production. Special properties and differences between the general formulation of the assimilation system and the production version, are outlined.

Chapter 3 deals with software engineering issues connected to the sequential and the planned parallel data assimilation algorithm. Constraints imposed or desired by the NASA Data Assimilation Office are collected. The analysis of the source code and the evaluation of the available resources show that many of the constraints are impossible to fulfill. The analysis motivates changing future project approaches by incooperating better software standards for the code development. One result from this analysis, is the desire to simplify the task of a programmer, which ultimately leads to the design of the meta-computing environment as introduced in Chapter 7.

In Chapter 4, possible parallel algorithms for the assimilation system are explored. Essential for the derivation of an efficient parallel algorithm, is the analysis of the physical domains and the data domains, on which the calculations are performed. Functional and data decomposition are employed at the same time in the presented algorithms. Different domain decompositions are analyzed. Regular, irregular, static and dynamic decompositions for the different domains are discussed. Future research for the other parallelization strategies are introduced. Task and data parallel algorithms are considered. Pointers to similar other parallel assimilation systems, as found in literature, are given.

Chapter 5 analyzes a problem inherent with the assimilation system, which was not considered previously. The problem is based on the nondeterminism of the quality control algorithm as used in the assimilation system. A deterministic algorithm is presented, which has been recently introduced to the NASA Data Assimilation Office, motivating the future development of a quality control algorithm based on the results presented here.

Chapter 6 presents performance data obtained with different decomposition algorithms, as introduced in Chapter 4. Timings for different sequential and parallel machines are depicted.

The second part of the dissertation, deals with the limited resources available to solve grand challenges. A metacomputing environment is designed, which simplifies the execution and program development on supercomputers.
In Chapter 7, a general concept of metaproblems is described. An essential part of the research conducted, dealt with resource limitations and finding solutions to circumvent them. Dealing with a grand challenge problem and the many different computing resources used for the program execution, motivates the design of a metacomputing environment.
Chapter 8 introduces the concept of the components of the planned metacomputing environment. Besides the development of efficient massively parallel programs to solve a grand challenge, the environment in which the program is executed should be transparent and intuitive for the user of heterogeneous supercomputing environments. The classification of loosely coupled and tightly coupled metacomputing environments is introduced, based on the granularity of the parallelism, embedded in the problem, to be solved. Design issues for a user interface are presented which fulfill both requirements: incooperating multiple programming paradigms and a programming paradigm based on the dataflow concept. Related research, actively pursued at many different institutions, are pointed out. The current state of a prototype implementation and further improvements are presented.
Chapter 9 presents the accomplished goals and conclusions of this dissertation. Further improvements for a parallel assimilation system, as well as, the metacomputing environment are discussed.

Chapter 2

Data Analysis in Atmospheric Science

2.1 Climate Modeling

The precise prediction of weather and climate is an essential part of our daily life. The state of the atmosphere changes every moment. Deriving hourly, daily, seasonal and long term forecasts, helps to prepare for the changes in the state of the atmosphere. The difference in the space and time scale phenomena of the earth climate system are of large variety (Figure 2.1[26]). On the smallest scale, turbulence is studied, while on the larger scale, changes in the climate and the CO_2 values are examined. The region of interest for weather and climate models ranges from about 10 km to 36,000 km, the circumference of the earth. The time scale of interest ranges from a few hours, to months, years, and for long running CO_2 analysis, even longer.

For all of them, it is desirable to derive methods which can predict the behavior of the atmosphere at the different scales. A precise forecast enables one to determine proper preventive actions governed by the future atmospheric condition. The different demands in space and time is also the reason for deriving different computational simulation models. Weather forecast models are used for medium scale and length atmospheric events, like the occurrence of tornados[15]. The results of the simulation are presented to us daily over numerous TV cable channels. Large scale climate models provide the opportunity to study long term at-

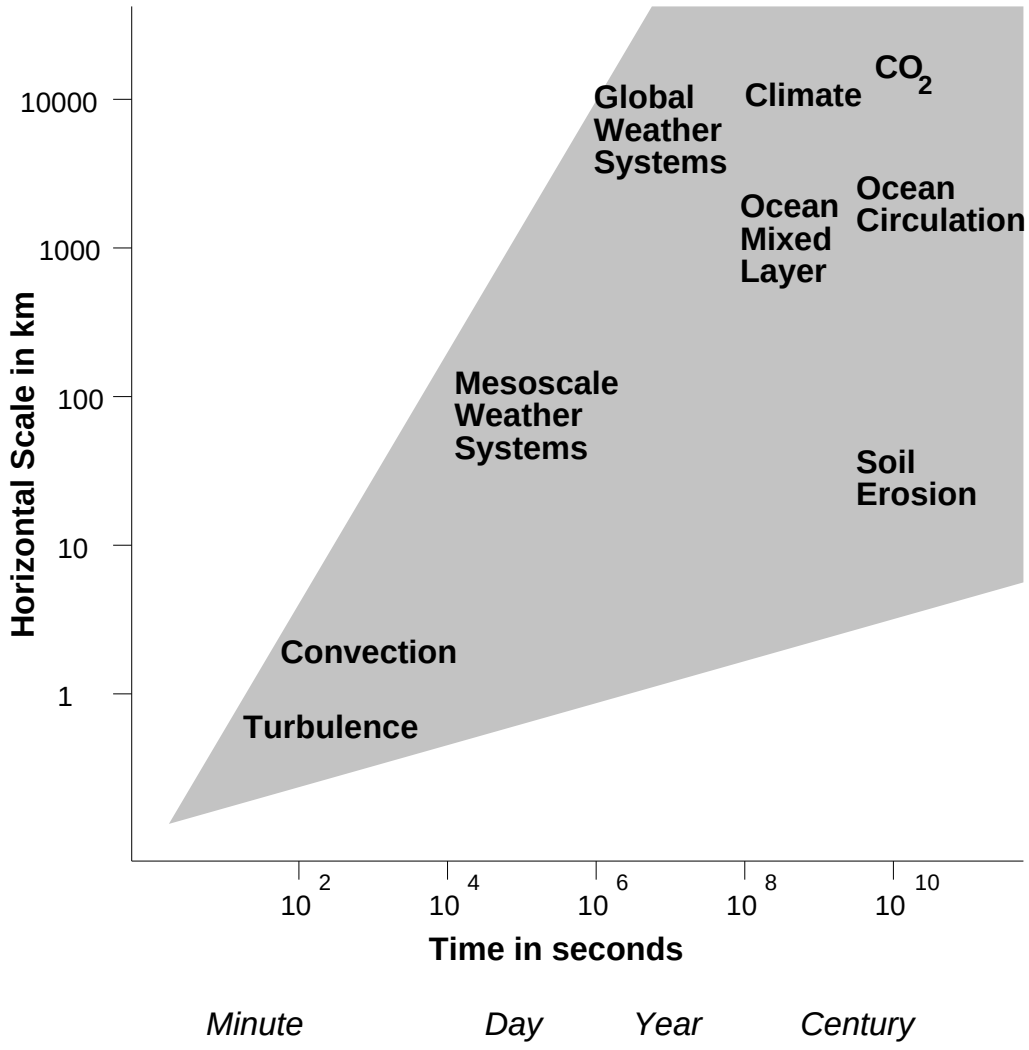

Figure 2.1: *Space and time scales of an earth climate system.*

mospheric events, like global warming[1]. Due to the similar nature of climate and weather forecast models, many overlaps exist between them. Often, weather forecast models are based on a limited and smaller scaled climate model simulation.

Unfortunately, the atmosphere is the most variable component of an earth climate system. Elaborate models are necessary to describe the behavior of the atmosphere[104, 107]. Not only do climate models contain complicated equations, but also perform calculations on large amounts of data sets. Thus, computers with enormous computational power and storage capacity are needed to calculate a prediction of only small complexity. Climate modeling is classified as one of the grand challenge problems[35] because of its scientific value and the amount of computational resources necessary to pursue the calculation (Figure 2.2). The calculation cannot be performed on any single existing computer in sufficient time. Due to the complexity of the calculation and the large storage volume, climate modeling is even one of the most challenging applications from the large scale grand challenge problems.

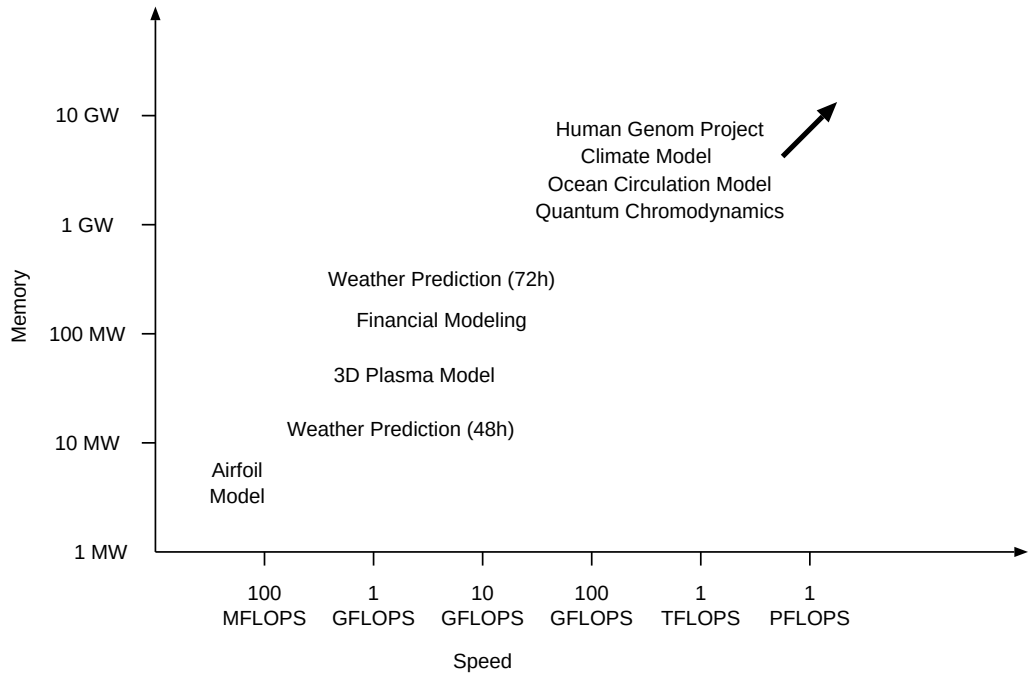

Figure 2.2: *The memory and speed requirements of some grand challenge problems in WORDS and FLOPS.*

2.2 Data Analysis

To clarify the role of data assimilation in relationship to climate modeling, a more detailed analysis of the field of meteorology is necessary.
The main problem in meteorology is how to obtain a valid forecast. Early in the development of the science of meteorology, three steps have been distinguished:

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

These three steps are influencing each other directly. The forecast will be unreliable without a precise initial state, even if the governing equations describing the atmosphere are as accurate as possible. The best initial data will be worthless, if the equations describing the atmospheric model are inaccurate. Furthermore, only small errors should be introduced while applying the mathematical evaluation process on the initial condition, with the help of the governing equations. Obviously, a *perfect* forecast of the atmosphere is not possible due to its chaotic nature. The goal is to be as precise as possible.
A major step towards an automated forecast system has been achieved by Richardson, in his ground breaking publication[91]. He used a finite difference form to integrate forward in time, obtaining a forecast from an initial state (originally published in [12]). The difference scheme is applied on a regular latitude-longitude grid of fixed size.
A set of 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. Variable values in the vertical are referred to as *profiles*.
Unfortunately, the initial data in Richardson's first forecast calculation was incomplete. The values of the field were not defined at all grid points of the domain. The forward integration can only be achieved after all values for the fields have been defined. The question arises:

> How are the initial values for the computational grid determined, when the actual value at the grid point is unknown?

A method had to be derived to fill in the missing values. Richardson analyzed the existing observation values subjectively and estimated the missing values at several grid points. This *subjective analysis* method was used for a long time, reflecting the fact that the experience of the scientists interpreting the data had a large subjective influence on the quality of the initial solution. Since then, the subjective analysis method has been replaced by numerical algorithms which can be executed by computers. In order to stress the fact that these methods are not anymore relying on the experience of a scientist, the term *objective analysis* is used. Unfortunately, the term objective is somewhat confusing. It does not mean that the method is perfect or without any subjectivity (it has been developed by scientists and represents a subjective method for solving the problem), but it does reflect the fact, that the subjective analysis method by a scientist is replaced by a computing device. In today's forecast systems, many values for the grid are still missing and have to be estimated, even with the increased number of observation devices available today.

To quantify this problem, a typical distribution of the locations of all available observations during a six hour time interval is shown in Figure 2.3. In this realistic example, regions exist with no available observation data, even though satellites, ships, and airplanes are used to obtain the data. A schematic closeup is shown on the left side of Figure 2.4. In this Figure, the values at the grid points are unknown. The observations are used to obtain the values. Many possible solutions for obtaining the initial values have been introduced in literature[26].

2.3 Optimal Interpolation

One of the most successful and often used objective analysis methods, is a technique called *optimal interpolation*. This technique was introduced by Eliassen[34] and Gandin[55]. 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 observations. Then, for each grid point, 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 an interpolation step from the fields towards the actual observation locations.

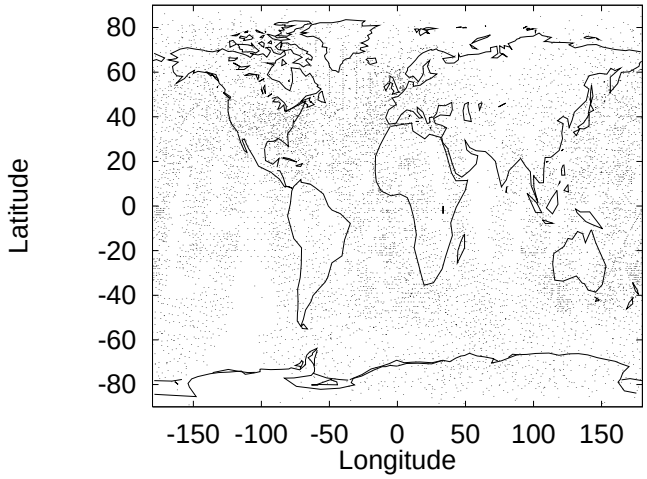

Figure 2.3: *Typical distribution of observations for the determination of an initial state.*

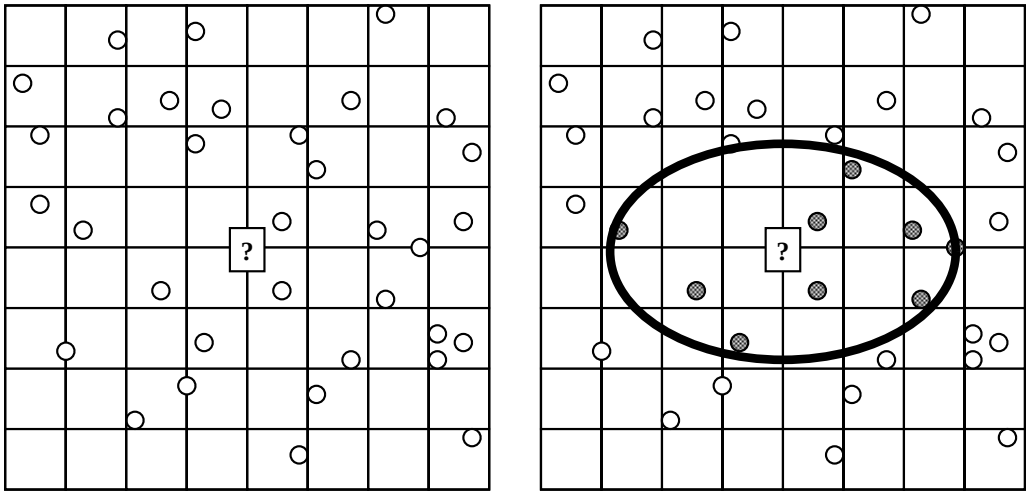

Figure 2.4: *Schematic closeup of a typical distribution of observations for the determination of an initial state. The area of influence is shown for the grid point in the middle.*

In practice, a *cut-off* distance is used allowing only observations up to a fixed distance to be included for the update. This is shown schematically on the right side of Figure 2.4. The cut-off strategy will be described in more detail in Section 2.6.5 and 4.6.2.
First, a simplified mathematical formulation of the optimal interpolation algorithm is given. The description follows the univariant case, which has the property that no correlation exists between the variables of one field to another field. Let,

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

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

$\mathbf{F_g}$ be the first guess value at the grid point 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.

Let, $\langle\ \rangle$ denote the statistical averaging process, and let, $\langle \epsilon_g^{\mathbf{A}} (\epsilon_g^{\mathbf{A}})^{\mathbf{T}} \rangle = \langle \epsilon_g (\epsilon_g)^{\mathbf{T}} \rangle^{\mathbf{A}} = \langle \epsilon_g \epsilon_g^{\mathbf{T}} \rangle^{\mathbf{A}}$, where $\epsilon_g^{\mathbf{A}}$ is the error at the grid point g for field A. Then, the optimal interpolation algorithm can be derived as follows:

$$\mathbf{A_g} = \mathbf{F_g} + \sum_{\mathbf{i=1}}^{\mathbf{N_g}} \mathbf{W_{gi}} (\mathbf{O_i} - \mathbf{F_i}) \ , \tag{2.1}$$

where $\mathbf{A_g} - \mathbf{F_g}$ specifies the *analysis increment* or *correction*, and $\mathbf{O_i} - \mathbf{F_i}$ specifies the *observation increment* or *innovation*. Transforming Equation (2.1) in terms of errors, yields to

$$\epsilon_g^{\mathbf{A}} = \epsilon_g^{\mathbf{F}} + \sum_{\mathbf{i=1}}^{\mathbf{N_g}} \mathbf{W_{gi}} (\epsilon_i^{\mathbf{O}} - \epsilon_i^{\mathbf{F}}) \tag{2.2}$$

Then,

$$\langle \epsilon_g (\epsilon_g)^{\mathbf{T}} \rangle^{\mathbf{A}} \tag{2.3}$$

can be minimized with respect to the weights,

$$\frac{\partial}{\partial \mathbf{W}} \langle \epsilon_{\mathbf{g}} (\epsilon_{\mathbf{g}})^{\mathbf{T}} \rangle^{\mathbf{A}} = \mathbf{0} \tag{2.4}$$

$$\langle \epsilon_{\mathbf{g}} \epsilon_{\mathbf{j}}^{\mathbf{T}} \rangle^{\mathbf{F}} = \sum_{\mathbf{i=1}}^{\mathbf{N_g}} \mathbf{W_{gi}} (\langle \epsilon_{\mathbf{i}} \epsilon_{\mathbf{j}}^{\mathbf{T}} \rangle^{\mathbf{F}} + \langle \epsilon_{\mathbf{i}} \epsilon_{\mathbf{j}}^{\mathbf{T}} \rangle^{\mathbf{O}}) \ . \tag{2.5}$$

In the Equation (2.5), 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:

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

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

where δ and ρ are functionals dependent on the observation positions, their difference, and the observed variable. An exact specification of these complex functions and their derivation can be found in [88, 26, 7]. Similarly, let, $\langle \epsilon_{\mathbf{g}} \epsilon_{\mathbf{j}}^{\mathbf{T}} \rangle^{\mathbf{F}}$ be approximated by

$$\langle \epsilon_{\mathbf{g}} \epsilon_{\mathbf{j}}^{\mathbf{T}} \rangle^{\mathbf{F}} \backsimeq \sigma_{\mathbf{g}}^{\mathbf{F}} \sigma_{\mathbf{j}}^{\mathbf{F}} \varrho(\mathbf{d})$$

Then, we obtain in a final form

$$\underbrace{\sigma_g^F \sigma_j^F \varrho(d)}_{b_g} = \sum_{i=1}^{N} \mathbf{W_{gi}} \quad \underbrace{(\sigma_i^F \sigma_j^F \delta_{ij}^F + (\sigma^2)^O \rho_{ij}^O)}_{\mathcal{A}_g} \tag{2.6}$$

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

$$\mathcal{A}_g x = b_g \ . \tag{2.7}$$

This is done with the help of a Cholesky factorization[56].[1] 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 (2.5) become matrices instead of vectors. For a complete derivation of a multivariate optimal interpolation algorithm, we refer to [26]. The extensive parameter set, as used in an operational OI algorithm, is described in [88].

2.4 Quality Control

Besides using the optimal interpolation strategy, it is of utmost importance to ensure the quality of the observations which are considered in the actual analysis. A network of various observation instruments gathers data about the atmospheric condition. Generally, the instruments are divided into three classes:

Instruments of class 1 measure observations taken at a single observation location. 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 radars measuring precipitation and winds via Doppler shift.

Instruments of class 3 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, and 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. Fortunately, the characteristics of an instrument and its error can be predetermined, on average, for each instrument type. In order to eliminate instrument errors, the characteristics are an important input of the actual data analysis. To avoid variations of

[1] The matrix is symmetric and positive definite

the initial data and the actual predicted atmospheric state, it is necessary to check the input observations and correct or reject erroneous data from the analysis. Certain errors can be corrected quite easily, leading to an overall improved quality of the analysis. In [26], details about the numerous errors of the different instruments can be found.

2.5 Applications of Data Analysis

Besides finding values to initialize a climate model, data analysis can be used in a much broader sense. It can be used to find and improve new and existing climate models, emphasizing the increased importance of this branch in atmospheric science. It starts with the collection of a sufficiently large data base of past observations on the earth. These observations are then transfered to a representation conforming with the model (atmospheric fields). Comparing the results obtained from the forecast and the data analysis, enables one to find differences between the model and the observations obtained in reality. The differences can be studied and used to improve the model such that a more reliable forecast will be achieved. In case a model has been verified as sufficiently accurate, a long term prognosis can be run. Providing the data and the assimilation system with an integrated climate model is one of the goals pursued at the NASA Data Assimilation Office (DAO), at Goddard Space Flight Center (GSFC)[29].

2.6 The Operational NASA Four Dimensional Data Assimilation System

2.6.1 The NASA Assimilation System

In this section, the specific details of the operational assimilation system are described, as used at DAO.

The assimilation system provides researchers with the ability to forecast the state of the atmosphere, based on a data set, collected over the past ten years. The NASA Four Dimensional Data Assimilation System(DAS) constitutes of independent program modules, as shown in Figure 2.5. First, data observed by satellites, radiosonds, weather balloons, airplanes, and many other sources are prepared for input. A quality control check is performed to eliminate

wrong or erroneous data.

After the quality control, the objective analysis is performed. The objective analysis – henceforth called *the analysis* – involves the 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. Then, the model calculation is performed[89]. A general circulation model (GCM) – henceforth called *the model* – is used to generate a six hour forecast.

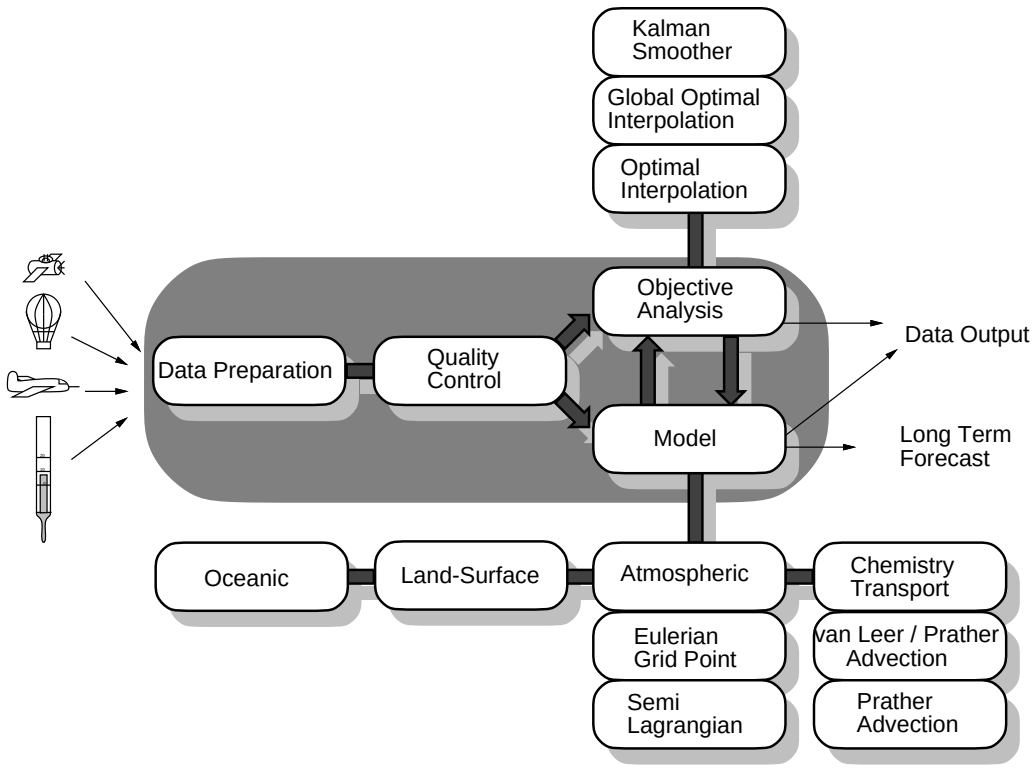

Figure 2.5: *Overview of the Integrated NASA Goddard Data Assimilation System.*

At the DAO, different strategies for the analysis are in use, as well as under development. For example, a multivariate optimal interpolation algorithm[95, 26], a global analysis algorithm, called the Physical-space Statistical Analysis System (PSAS)[98, 57], and the Kalman Filter[23, 80], are among the data assimilation strategies. Currently, a method called optimal interpolation algorithm (OI) is used as part of the operational integrated data assimilation system[7].

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.[2] Data is interpolated from non-uniform observation locations to a regular latitude-longitude grid via the multivariate optimum interpolation (OI) analysis technique[89]. The OI algorithm uses statistical estimates to determine appropriate relative weighting between *noisy* observations and a somewhat inaccurate first guess, obtained with a forecast by the model. This is done in order to minimize the resulting error in the analysis and forecast.
The DAS 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 on the data and fields available at the time (Figure 2.6). The state of the fields from the past, present, and future, are important for the calculation and evaluation.
The process of iterating and obtaining the results for the calculation is referred to in literature as *Four Dimensional Data Assimilation.* Three dimensions represent the physical space, while the forth dimension is given by the time. The method performing the objective analysis (the data analysis) is called *Objective Assimilation System*, or simply the *Assimilation System.* The assimilation system refers only to the data preparation, the quality control and the objective analysis method. The model is not included.

2.6.2 Model Resolution

A problematic issue in climate modeling, is the resolution used for the grid representation. In case the grid is dimensioned wrong, an error based on its representation is introduced. This error is referred to as the *error of representativeness.* If the scale of the grid is too big, effects, such as the well known *lake effect snow* in Syracuse, would not be detected. Therefore, it is desirable to design a model which contains a high number of grid points over the earth. This concludes that grids with big gaps between the grid points, introduce a high error in the

[2]For current and near future demands, 100,000 observations are considered.

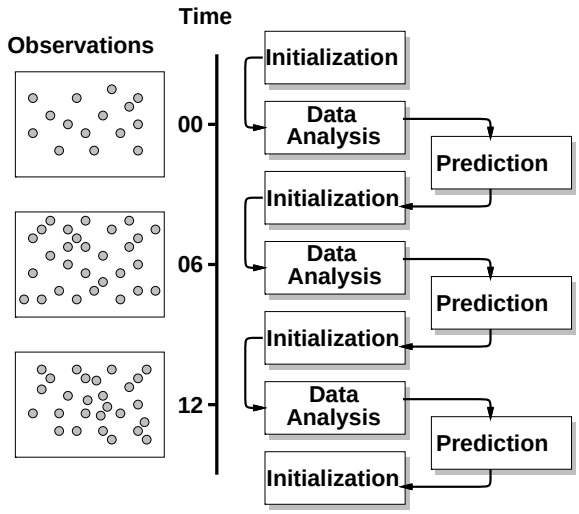

Figure 2.6: *The data assimilation cycle.*

calculation. On the other hand, the calculation performed on a computer will be much less time consuming with a smaller grid. A careful selection has to be made to choose the correct grid size and the desired resolution accuracy. They are inversely proportional to each other:

$$accuracy \sim gridsize \sim \frac{1}{resolution}$$

The resolution of the grid can be chosen arbitrarily in the production code. Usually resolutions of $2 \times 2.5 \times 20$, $2 \times 2.5 \times 46$, or $4 \times 5 \times 20$ are used, because these grids are compromises between speed and accuracy, and are used at other institutes to achieve comparable results. The first two numbers reflect the distance, in degrees, between a grid point in longitude and latitude direction. The last number reflects the number of horizontal levels in the field. Therefore, one can obtain a grid of about $90 \times 144 \times 20$, $90 \times 144 \times 46$, and $45 \times 72 \times 20$, respectively. In the NASA code, the vertical levels correspond to the pressure measured in a σ-coordinate model. In this coordinate representation, the different height levels are parallel to the surface level. In contrast to a z-coordinate model, where the height is specified by the geometric position.

The grid representation for the analysis and the model is different. Currently, the objective analysis based on the OI uses σ coordinates, while the model uses z-coordinates. Special

routines allow the transformation of the field variables between the different representations.

2.6.3 The NASA Optimal Interpolation Algorithm

The OI algorithm used in the data assimilation system is split in three separate modules, as depicted in Figure 2.7. The surface-level-pressure (SLP) analysis, the moisture-vapor (MIX) analysis and the height-u-wind-v-wind (HUV) analysis are differentiated. 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. The HUV and MIX analysis modules have the same program flow as the SLP analysis.

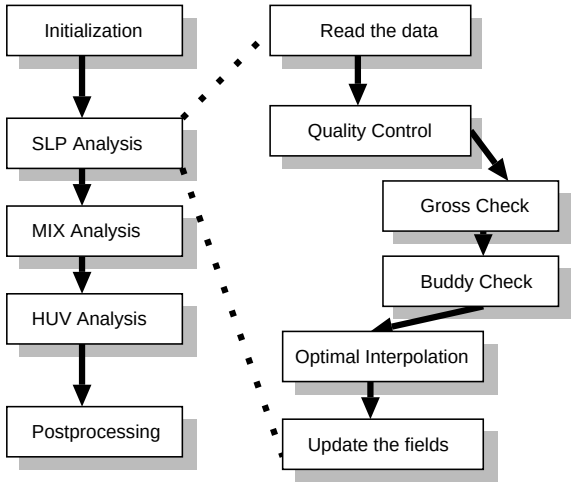

Figure 2.7: *The modules of the OI algorithm. The details of the SLP analysis are shown. The MIX and HUV have the same functional decomposition as the SLP analysis.*

Without loss of generality, only the SLP analysis is shown on the right side of Figure 2.7. While the SLP analysis is done on a two dimensional domain, HUV and MIX are performed in the three dimensional domain. The computational demand for the HUV analysis is the largest due to the amount of data influencing the big three dimensional domain. Since there is only few data available for the MIX analysis, the time spent for this analysis is the smallest. In order to specify a parallel algorithm for the optimal interpolation, it is first necessary to describe the function of the separate modules, with the help of high level program descriptions (Figures 2.2 - 2.5). Each of the different analysis parts consists of a

1. **data input** routine, to read the necessary observations in order to perform an accurate prediction.

2. **quality control** algorithm, to exclude and correct erroneous observation data.

3. **optimal interpolation** algorithm, to perform the calculation given in Equation 2.7 for each grid point.

The resulting high level description is displayed in Program 2.1.

Program 2.1 The objective analysis algorithm of the operational NASA data assimilation system.

```
1  proc Objective Analysis based on OI
2        – SLP analysis:
3        Read in the observations for (SLP)
4        Perform Quality Control (SLP)
5        Perform Optimal Interpolation (SLP)
6        – MIX analysis:
7        Read in the observations for (MIX)
8        Perform Quality Control (MIX)
9        Perform Optimal Interpolation (MIX)
10       – HUV analysis:
11       Read in the observations for (HUV)
12       Perform Quality Control (HUV)
13       Perform Optimal Interpolation (HUV)
14 end proc
```

2.6.4 Quality Control

First, the *quality control* algorithm is considered. The quality control is an essential part of the assimilation system at NASA. The internal strategy of the quality control uses a simple averaging process, similar to the derivation of the weights for the optimal interpolation. Program 2.2 describes the program flow for the quality control of the observation data. In line 7 and 9, 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 (Line 11 and 12).

Program 2.2 The quality control driver algorithm.

```
1  proc Quality Control
2        foreach observation do
3          if observation has invalid location
4            then mark observation as invalid
5          end if
6        end foreach
7         Obtain the first guess values at the location
8             of the valid observations.
9         Interpolate the forecast errors to the
10            observation locations.
11       call Gross Check
12       call Buddy Check
13 end proc
```

Program 2.3 The gross check algorithm.

```
1  proc Gross Check
2       T_h ← Tolerance
3       foreach valid observation do
4         if variable is not in allowed range
5           then mark observation as invalid
6         end if
7         if observation is (still) valid
8           then δ ← variable^O − variable^F
9                if δ^2 > T_h((σ^O)^2 + (σ^F)^2)
10                 then fail, mark observation as suspect
11               end if
12        end if
13      end foreach
14 end proc
```

Program 2.4 The buddy check algorithm.

```
1  proc Buddy Check
2       foreach suspicious observation do
3           Search for the set S of all valid observations in a particular
4            radius arround the data point.
5          analyzed value ← Perform univariate sucessive correction
6           method at the location of the observation with the help of S
7          δ ← observation − analyzed value
8          if δ is in tolerance level, δ ≤ ||T_h||
9            then mark observation as valid
10         end if
11      end foreach
12 end proc
```

The *gross-check* eliminates obvious errors, dependent only on the values at the location of the observation. A test is performed, based on the observation error variances σ^O and forecast error variances σ^F:

$$\delta^2 > T_h((\sigma^O)^2 + (\sigma^F)^2) \ , \tag{2.8}$$

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 description of the gross check is shown in Program 2.3.

The *buddy-check* eliminates errors which depend on the values of all the observations in the vicinity of the location. A univariate successive-correction method performs the analysis at each location, specified by a suspicious observation. Then, the difference between the suspect observation and the analyzed value are subjected to Equation (2.8). The observation is accepted if Equation (2.8) is not satisfied[95].

2.6.5 Optimal Interpolation

After the quality control is completed, the optimum interpolation algorithm is applied and the matrices, as given in Equation (2.7), are solved (Program 2.5). The term *matrix solve* is sometimes used alternatively to the term *optimal interpolation* to stress the fact, that it can

be replaced by another solving method, different from the optimal interpolation.

Program 2.5 Optimal interpolation algorithm

```
proc Optimal interpolation algorithm
    foreach gridpoint g
        Collect the data in the vicinity of this gridpoint
        foreach Vertical level
            Calculate covariance matrix A_g
            Solve the linear equation A_g x = b_g
            Store the result in the fields
        end foreach
    end foreach
end proc
```

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

To limit the size of the covariance matrices for the Cholesky factorization, a cutoff radius of about 1600km is used. In addition, there is an upper bound of 75 observations for the number of observations considered in the factorization. A sophisticated rule based decision process, which in turn is dependent on empirical data, decides which observations are used, in case more than 75 observations are in a region of influence.

2.6.6 Minivolume Concept

One very important difference between the optimal interpolation algorithm, as specified in Section 2.3, and the operational system, is the use of so called *minivolumes*. In the original formulation of the algorithm, the covariance matrices are computed for each grid point. Since the difference in the covariance matrices of one grid point and the neighboring grid points is rather small, the covariance matrices are determined for a group of grid points, laying in a small latitude-longitude-height(pressure) box. Only when the geographical distance between the gridpoints is large, substantial differences in the covariance matrix occur. The term *minivolume optimal interpolation algorithm* is used to describe this algorithm. Since the setup

of the covariance matrix is rather time consuming, the introduction of minivolumes decreases the calculation speed drastically. Unfortunately, it reduces the quality of the solution in contrast to the original algorithm[102].

The current setup has the minivolumes distributed in the way as depicted in Figure 2.8. The reason for more minivolumes in the low latitude bands is based on the fact that the Cartesian latitude-longitude coordinate space on the earth does not preserve the property of equal distance (see Section 4.4 for more detail). If one maps the Cartesian coordinate system to a spherical coordinate system, dense regions of coordinate grid points would exist around the pole. In this case, the calculation would be unnecessarily slowed down without any precision gain. The choice of the coordinate system is defined by DAO. It has the property that it generates high quality solutions, in a reasonable time. The speed up for setting up the covariance matrices of the minivolumes, distributed as shown in Figure 2.8, and the gridpoint is as follows:

Let, l denote the number of levels in the grid domain. Then, the number of minivolumes used in the operational system is currently $1046 \times \lceil \frac{l}{2} \rceil$. In the gridpoint OI algorithm, the number of covariance matrix generation is either, about $144 \times 90 \times l$ or about $72 \times 45 \times l$. This gives a speedup of about 24 for the larger grid, and 12 for the smaller, to setup the covariance matrices.

The outline of the minivolume based optimal interpolation algorithm is shown in Program 2.6.

Program 2.6 The minivolume based optimal interpolation algorithm

```
proc Minivolume based optimal interpolation algorithm
    foreach minivolume m
        Collect the data in the vicinity of this minivolume
      foreach Vertical level
        Calculate covariance matrix A_m
        foreach gridpoint g ∈ current minivolume m
          Solve the linear equation A_m x = b_g
          Store the result
        end foreach
      end foreach
    end foreach
end proc
```

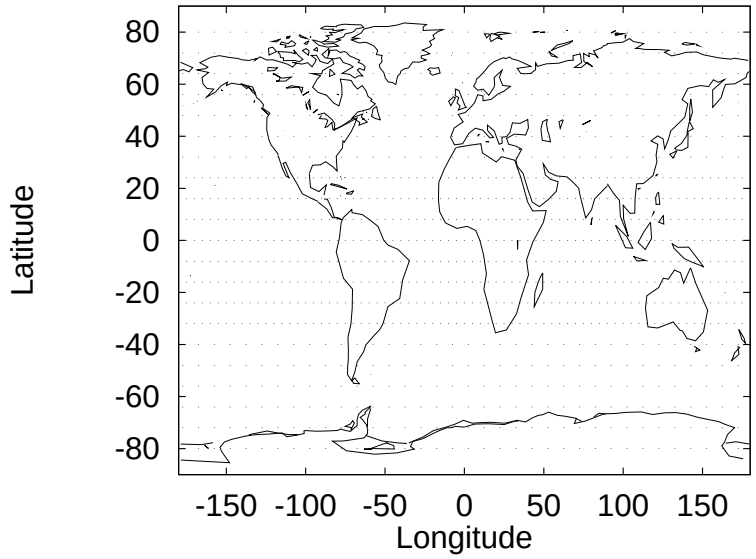

Figure 2.8: *Center of the minivolumes for the sample problem.*

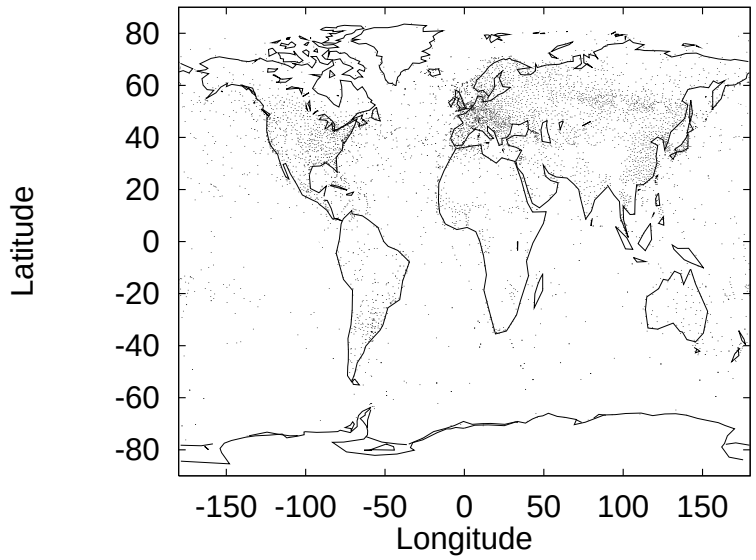

Figure 2.9: *Typical distribution of sea level observations as used for the objective analysis.*

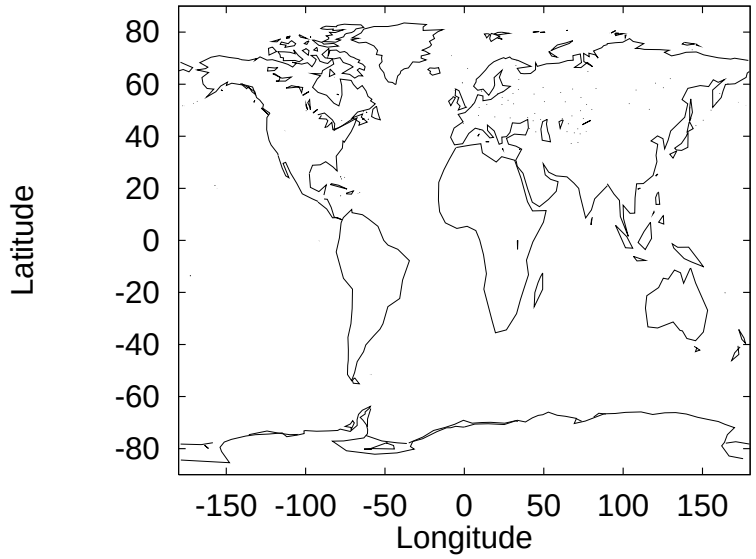

Figure 2.10: *Typical distribution of moisture observations as used for the objective analysis.*

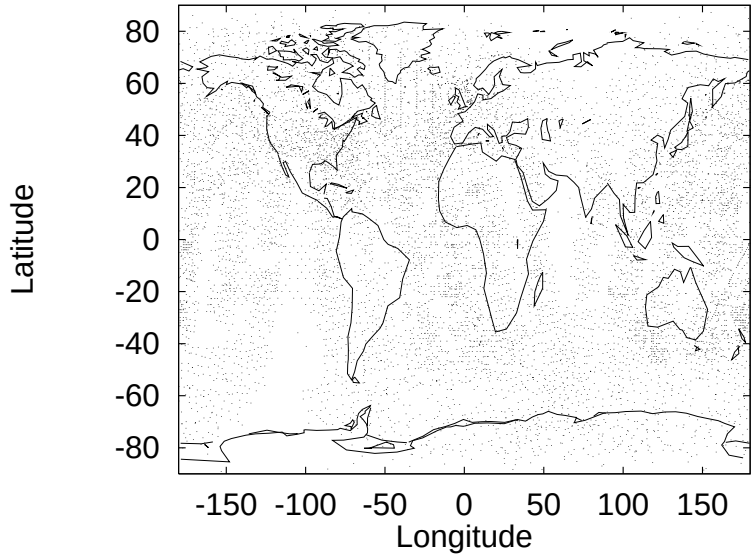

Figure 2.11: *Typical distribution of HUV observations as used for the objective analysis.*

2.6.7 Data Inputs

Besides the specification of the OI algorithm, it is important to know more about the properties of the data sets on which the algorithm operates. For many scientific applications a correlation exists between the density of the data and the time for the calculation for a subregion of the domain. This is the reason for load imbalance in many applications.
The distribution of the observations for the SLP, MIX, and HUV analysis are shown in Figures 2.9-2.11. The data for the SLP analysis consists of observations taken from the surface of the globe, including weather stations, ships, and others. The data for the MIX analysis consists of all data useful for the determination of the moisture-vapor calculation. Presently, there is not much moisture-vapor data available, but with improvements in technology and additional measurements, more data is expected soon. The HUV analysis uses data from satellites, balloons, airplanes and other sources, to obtain the fields in different horizontal layers. From the figures, it is clear that the observation data is quite differently distributed. Naturally, one expects more observations on the continents at sea-level pressure, due to the availability of many observation stations over land. Big regional gaps in the HUV analysis data are apparent because there is simply no satellite covering this particular area, or a satellite has not passed over the region in the 6 hour interval. With the start of new satellites at the turn of the century, this situation will be improved.

2.6.8 Incremental Analysis Update

A unique feature of the analysis system, as implemented at NASA, is the use of an incremental update algorithm. As in other assimilation systems, an analysis is performed every 6 hours. Using an incremental update, the analysis increments are not directly added to the first guess field. A model integration is restarted 3 hours prior to the analysis time. The integration is performed over the next six hours using the analysis increment added as a constant forcing term in the model equations. Then, the integration is extended for the next 3 hours to provide a first guess for the next analysis time. The incremental update improves the analysis, e.g., in terms of accuracy and noise control.
A useful term to describe the time at which observation values are taken, are the so called *synoptic times*. Surface and radiosond observations are taken regularly at synoptic times[26]. They are 00 and 12 GMT (Greenwich mean time) for radiosonds, and 00, 03, ..., 21, 24

GMT for surface observations. Satellite observations are recorded continuously. They are *asynoptic*. The incremental update algorithm makes use of the fact that a more up to date observation data set is available during the calculation.

Chapter 3

Imposed Constraints for the Parallelization of the Assimilation System

Usually, imposed constraints on the problem and program have to be fulfilled while parallelizing grand challenge applications. These constraints are often determined by the development team in order to guarantee the maintenance of the code in the future. Thus, besides the development of a theoretically sound algorithm, practical constraints have to be considered during the program design phase. This is especially true for atmospheric science applications, where only an actual running and maintainable code will be useful.
The following constraints were imposed by the DAO for a parallel data assimilation algorithm:

1. **Structural Equivalence** should be maintained. The parallelization should be done without major code restructuring and changes in the algorithmic flow.

2. **Correctness** of the program should be tested. This is done via exact numerical comparisons between the outputs generated by the sequential algorithms on a Cray supercomputer and the parallel target machine running the parallel code. This includes the reproduction of the results on the same input data.

3. **Portability** of the parallel code. The code should be run on many machines to make it available to a larger community. Furthermore, ANSI Fortran 77 is the programming

language for the parallel code.

4. **Simplicity** of the solution. The resulting algorithm should be easy to understand.

5. **Target machine Constraints**. The code should run on MIMD machines. If possible, it should also include SIMD machines.

6. **Ergonomic Constraints**

   (a) **Linear speedup** should be achieved. While using more processors, a near linear speedup should be obtained to make the algorithm scalable for machines with larger numbers of processors.

   (b) **Minimization of Development Cost/Time**. The code should be developed as quickly as possible.

   (c) **Minimization of the maintenance Cost/Time**. The code should have low maintenance costs.

In general, the constraints of simplicity, portability, and correctness are advantages for many projects. Achieving linear speedup can be an essential factor for the actual usability of the code on larger parallel machines.
Nevertheless, the constraint of the structural equivalence between the sequential and the parallel code can hinder the development of better, but structurally different, correct parallel algorithms. In the case of the objective analysis, it was chosen because of the limited manpower available to develop the code. The programming language Fortran is used for the code development for three reasons. First, the development team at DAO is familiar with Fortran. Second, the speed of Fortran is assumed to be superior to other languages in the field of atmospheric science, e. g. C and C++. Third, ANSI Fortran77 is available on most computers, thus enabling portability.
Because, the constraints were not carefully chosen, they were impossible to fulfill. This was mainly due to the design of the original sequential code. One major practical problem in the parallelization was the internal structure and software quality of the existing code.
The constraint to use ANSI Fortran77 was violated by the DAO itself, because the sequential code was not even written in ANSI Fortran77. It used vector constructs, dynamical allocated arrays and compiler directives for Cray Fortran77 compilers. Instead, ANSI Fortran90 has

been accepted as the programming language of choice, making it possible to use the code fragments which rely on vector constructs and dynamically declared arrays.
One of the biggest problems was to prove, that the result of sequential algorithm was dependent on the order of the observations. This would make a byte by byte comparison of the results, obtained from a parallel and the sequential algorithm, impossible. Chapter 5 explains why the algorithms behave differently. Thus, the difference between the fields calculated with different algorithms, should be as small as possible.

3.1 Software Engineering Problems

Programs used in atmospheric science are often very complex. Usually, they are of substantial size and have evolved over a number of years. Many domain specialists are responsible for the program generation and code maintenance. These specialists normally do not apply software engineering or software quality assurance practices. Thus, many codes are very difficult to handle and maintain[62]. The assimilation system code is no exception. As the following analysis will show, the code is worse than other codes known from atmospheric science. Consequently, future codes should be developed in a more strict and controlled environment. This is necessary because the parallelization of the sequential codes will even increase the complexity, as well as, the maintenance problems. Besides the validation of the program correctness, a substantial amount of documentation should be provided to ease future maintenance. Even though this appears as an additional cost in the beginning, it will pay off in future.
Because of its long history and the large number of programmers involved in the algorithmic development, the data assimilation system is very complex and can be considered as a *legacy code*. Thus, the sequential code was quite complicated, and a parallelization would take a considerable amount of time due to lack of documentation.
Another major problem, was the actual development stage of the sequential code. The first version of the code was buggy. While porting the sequential code with parallel extensions on different MIMD machines, many bugs in the original program could be revealed, thus, having implications on the improvement of the the actual sequential production version of the code.
A big problem for the portability of the code was the data input and output. The data handling was designed only for one specific computer platform, making use of internal features

of the filesystem. Driver routines, which allowed testing and program verification were not available. The original (modified) version of the code worked on ASCII data, which was completely insufficient for the code development due to size of the data and the time spent in the initialization.
As an immediate result from the research conducted, changes in the program development and more strict software engineering methods are currently under implementation at DAO, to prevent problems while parallelizing future codes. The DAO has developed new standards for writing FORTRAN code and started implementing a software quality control process for the validation of the program correctness[101], as a result of this analysis.

3.2 Software Metric Analysis

The following software metric analysis is especially directed towards coding practices often found in atmospheric science. A simple way of comparing codes is given.
Formally, computer software can be characterized by internal and external characteristics[37]. The internal characteristics include the size and the control flow complexity. The external characteristics contain measurements for the maintenance of the code by other researchers. It is extremely difficult to evaluate the characteristics of a code and compare them with other existing codes. Nevertheless, a general idea can be obtained for the control flow complexity while using measurements used by other researchers in the atmospheric science community. Such measurements, specifically used for the FORTRAN atmospheric codes[62], are displayed in the Figures 3.1-3.4. The following software metrics are distinguished:

Data coupling l – is the number of variables used in a particular subroutine. $l = f + g$, where l is the measurement for datacoupling, f is the number of variables in the COMMON blocks (global variables) for the routine, and g is the number of arguments to the function or subroutine.

Control coupling l' – is the count of the number of calls to other subroutines, added to the number of references to external user defined functions.

Size S – is a measurement for the size of the subroutine. It contains augmented counts of conditional statements, iterations loops, and jumps (RETURN, GO TOS). $S =$

$a + 5(b + c + 2d)$, where a is the number of uncommented lines, b is the number of conditions, c is the number of iteration loops, and d is the number of jumps.

Control flow complexity F – evaluates the number of binary decisions in a subroutine[36].

Naturally, the metrics only indicate a problem when its value is high in contrast to a comparable code. The code in question might be more *complex* than necessary[62].
Figures 3.1-3.4 contain the values for the codes from ECMWF, CSRIO, and BEST [74]. In addition, the values for the major subroutines of the analysis system codes are marked. These routines are similar in function to routines found in the other codes[101]. The following routines from the DAO code are displayed:

glassim is the main driver routine which includes many more subroutines than depicted in Program 2.1.

zuvcov is the generation procedure for the covariance matrices (Section 2.3) for the HUV analysis

solvdr is the routine which solves the matrices and corresponds to the high level description of the optimal interpolation algorithm (Program 2.6).

zuvanl is the quality control for the HUV analysis (Program 2.2).

It is clear that the DAO code is considerably more complex in size, control coupling, and control flow complexity. In the case of data coupling, it is comparable with the other programs. Unfortunately, there is not sufficient data currently available to compare the external characteristics of the codes. In comparison with the CAPS storm prediction code[115, 15], the DAO code rates far below. The CAPS code documentation is available for each of the routines. The sequential parts of the program have also been redesigned in consideration of the implementation on MIMD and SIMD supercomputers[93]. The DAO did not encourage a complete program redesign, due to lack of manpower. The lessons learned from the parallelization of the DAO code, and its poor characteristics, motivated a change towards the complete new development of an alternative to the current objective analysis code. The new code will produce better results in comparison to the OI, but it will be slower[57]. Table 3.1 summarizes some properties of the optimal interpolation and quality control code, as used in the current objective assimilation scheme. This analysis is typical for many scientific codes.

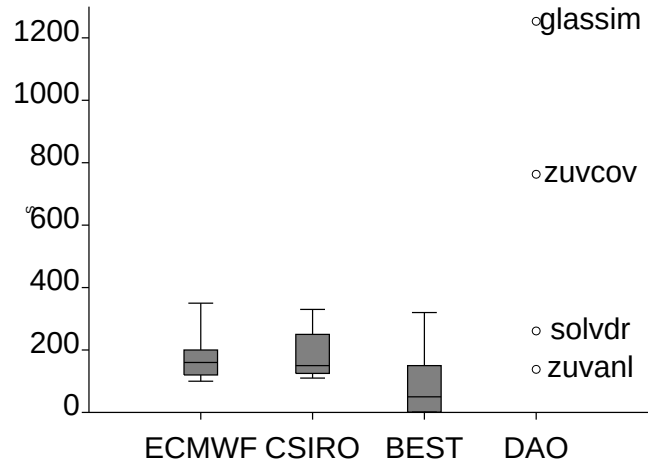

Figure 3.1: *Comparison of the software metric* size *S of the different programs*

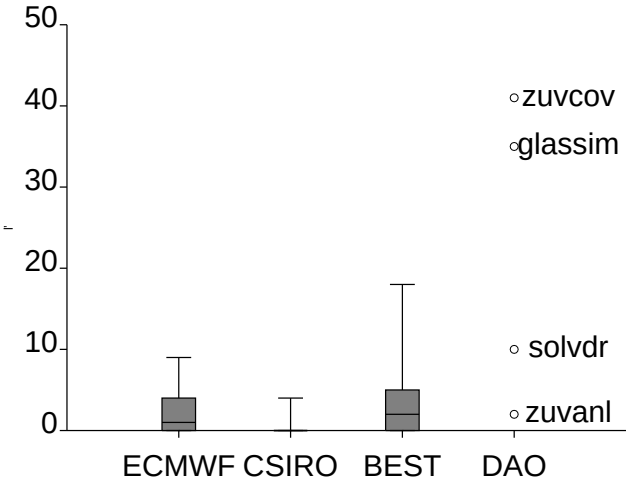

Figure 3.2: *Comparison of the software metric* control coupling *l of the different programs (number of calls and external references)*

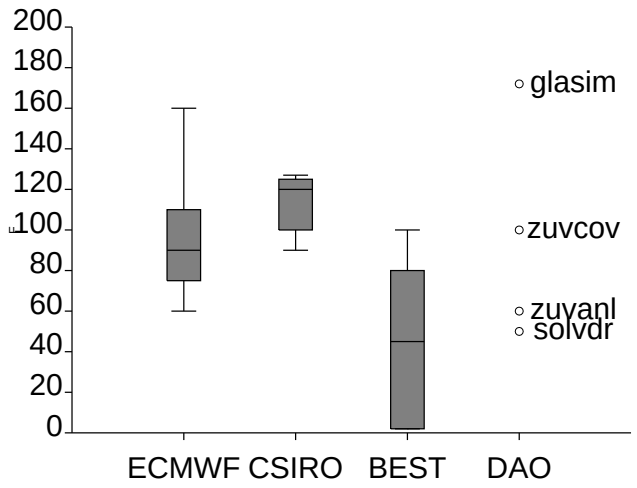

Figure 3.3: *Comparison of the software metric* data coupling *l' of the different programs (common block variables + calling arguments)*

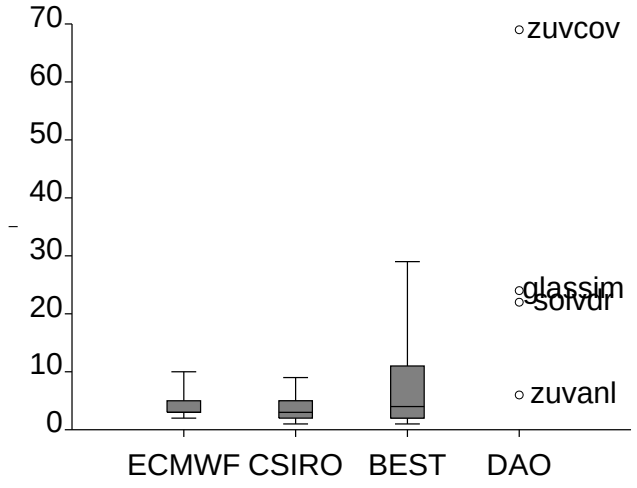

Figure 3.4: *Comparison of the software metric* control flow *F of the different programs (conditions, loops, and jumps)*

Table 3.1: Code Analysis of the Optimal Interpolation Algorithm as used in Four-Dimensional Data Assimilation

	Grand Challenge (NASA)
Legacy code	yes
Number of programmers	very high
Development cost	high
Number of Lines	69000
	(large)
Input/Output	not portable
Code quality	poor
Documentation	poor
Programming paradigm	sequential
Programming Language	Fortran
Computational load	high
Data volume	high
wall-clock time important	very
Machine precision	high
Parallelization potential	vectorization: medium MIMD : high

3.3 Software Engineering Choices

Some of the above established requirements can be overcome while using appropriate software engineering tools. For the parallelization, the following tools were useful:

Fortran90 and HPF: For the computational core of the main program, Fortran90 is used. In addition, it will be possible to transfer major parts of the Fortran90 program to HPF once the compilers are more stable.

C/C++: C/C++ is used for some of the data redistribution tools, because data abstraction and the development of software libraries are easier in an object oriented programming language[78]. 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++. No performance loss is expected for the parallel algorithm.

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 in Fortran90 to fulfill the requirement of a pure Fortran program.

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

Message Passing: For the intercommunication library, the standard Message Passing Interface (MPI) is used[82]. 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[9, 10] is easily possible while replacing the elementary message passing calls. Due to the design and extensions of the message passing libraries used, the use of heterogeneous computing environments is also possible. MPI will provide an efficient interface to supercomputers.

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

Chapter 4

A Parallel Objective Analysis System

4.1 The Parallel Programming Models

The program analysis from Chapter 3 is an essential part of the definition of a parallel program for the existing sequential data analysis system. The constraints and requirements have a great influence in the definition and specification of a final parallel code.
Ultimately, it would be desirable to obtain a fast code running on as many parallel machines as possible. Thus, it is useful to consider a parallel abstraction model, which is supported on as many machines as possible.
The abstraction models for most of the existing parallel machines are either based on the message passing model or the data parallel model[47, 3]. In a message passing program, parallelism is expressed explicitly with the help of communicating processes. Each process works independently and can exchange messages with other processes. In a data parallel program, parallelism is expressed implicitly. The data is divided amongst the processors. A program working on the data is assigned to each processor. In contrast to the message passing model, the data parallel model supports the view of a *global* array, where each array element can be accessed by each processor. Using global arrays can make the parallel programming task much easier.
The introduction of High Performance Fortran (HPF) allows one to simplify the program development on architecturally different machines. While SIMD machines support data parallelism, MIMD machines primarily support message passing paradigms. HPF provides the opportunity to use standardized data parallel programming constructs on MIMD and SIMD

machines. Hence, a HPF program will run on many different machines. An evaluation of the performance and usability of existent HPF compilers, at the beginning of the parallelization of the analysis system, showed that they would not be mature enough to pursue the project. Thus, the development of a message passing parallel algorithm is justified. An experimental HPF implementation of the quality control algorithm is described in[81]. Today, HPF compilers are more stable. They can be used for larger projects[59] and are a viable alternative to message passing programs for many problems.
One of the research goals conducted was to find out if a program representation can be found which supports the message passing, but also, the data parallel programming paradigm. This is important to guarantee the portability of the algorithm on actual supercomputers, which usually have only a limited life span. The final parallel data analysis program can support both paradigms, since it is developed with both programming models in mind.
In this Chapter, the domain and functional decompositions are introduced. Obtaining good domain decompositions, as well as, good functional decompositions of the program, is the key issue of specifying an efficient parallel algorithm. Once the domains and the functions performing the calculations on the domains are determined, a message passing parallel program can be derived.

4.2 The Data Domains of the Analysis System

First, it is necessary to distinguish between the *physical* domain and the *spatial* domain. For our purpose, the physical domain is represented by the earth and the atmosphere around it. The spatial domain is represented by the locations on the earth, on which a numerical calculation is performed. *Data domains* represent the input and output data used for a calculation in the spatial domain.
Thus, for the OI algorithm four data domains are distinguished:

Model variable domain – The model variables define the state of the atmosphere. In atmospheric science, the model variables are referred to as *fields*. The domain is specified by a longitude-latitude grid.

Grid domain – The grid domain specifies the locations on which an *interpolation* is performed. For the data assimilation system, each gridpoint has a set of model variables

which are stored in the model variable domain.

Observational data – The observational data is gathered before each analysis step is invoked. The domain is specified by coordinates in the latitude longitude grid, as well as, a height (pressure) coordinate.

Minivolume domain – In the case of the minivolume OI, the determination of the covariance matrices are only performed at the location of the center of the minivolumes. In the case of the grid point OI, the covariance matrix is determined for each gridpoint.

For the parallelization of a program, the identification of data domains and their mapping onto a parallel machine is necessary. It is helpful to know which properties are connected with the data, distributed over the domain. Often, *regular* and *irregular* distributions of data over the spatial domain are distinguished. In case the data is regularly distributed, it is sometimes possible to find straight forward parallel algorithms, which perform at times even efficiently.
Thus, the data domains of the analysis system are first specified and their properties in different coordinate systems are analyzed. This is motivated by the hope to find a regular representation, which leads to an efficient parallel implementation.

4.3 Loadbalance and Databalance

To establish a simple way of describing and evaluating the different domain decompositions, the following simple abstraction model of a parallel computer is used. The model is based on a set of $P = \{p_1, ..., p_n\}$ processors. A set of data $D = \{d_1, ..., d_m\}$ is distributed onto the processors in an exclusive way, where $m \gg n$. The size of each data item can be different.
The time of a calculation on each processor depend on:

1. the array index of the data in the data set (for the DAS, this is the geographical location).
2. the amount of data assigned to a processor.
3. the value of the data.
4. the data assigned to neighboring processors.

In parallel computing, it is essential to obtain the results of the calculation on all processors at approximately the same time. The calculation is *load balanced*. Loadbalance ensures high efficiency. In many applications, the domain decomposition is the key factor towards obtaining loadbalance.
Many times loadbalance can be achieved while mapping an equal amount of data to each processor and performing the calculations associated with the data. This is the case, when the calculation does not depend on the value of the data. It can help to specify a coordinate system in which the data is regularly distributed and domain decompositions are easy to find. Furthermore, it should be determined whether a coordinate system can be found, which uses significantly less space for a representation of the physical domain, in order to save memory. Time might be saved while performing fewer calculations on the smaller grid domain. It has to be assured that the accuracy of the representation of the physical domain does not suffer while using a smaller domain.

4.4 Coordinate Systems and Data Domains

4.4.1 Coordinate Systems based on Model Variable and Grid Domain

In the DAS, the model variables are used in the optimal interpolation and the quality control algorithm. Examples for different mappings from the physical domain onto different spatial domains are displayed in Figure 4.1. In the case of the OI, the calculations are performed on a grid, hence, the term *grid point* domain is used equivalently to the term spatial domain. The properties of the underlying coordinate system in which the calculation is performed, has an impact on the regularity of the distribution. To show this fact, four physical domain representations are compared: a Cartesian latitude-longitude coordinate system, an irregular latitude-longitude coordinate system, a spherical coordinate system, and an icosahedral coordinate system.

Cartesian Latitude-Longitude Coordinate System

Many climate models are defined in a Cartesian latitude-longitude coordinate system. A regular mesh, in the form of a grid, is embedded onto the surface of the earth.

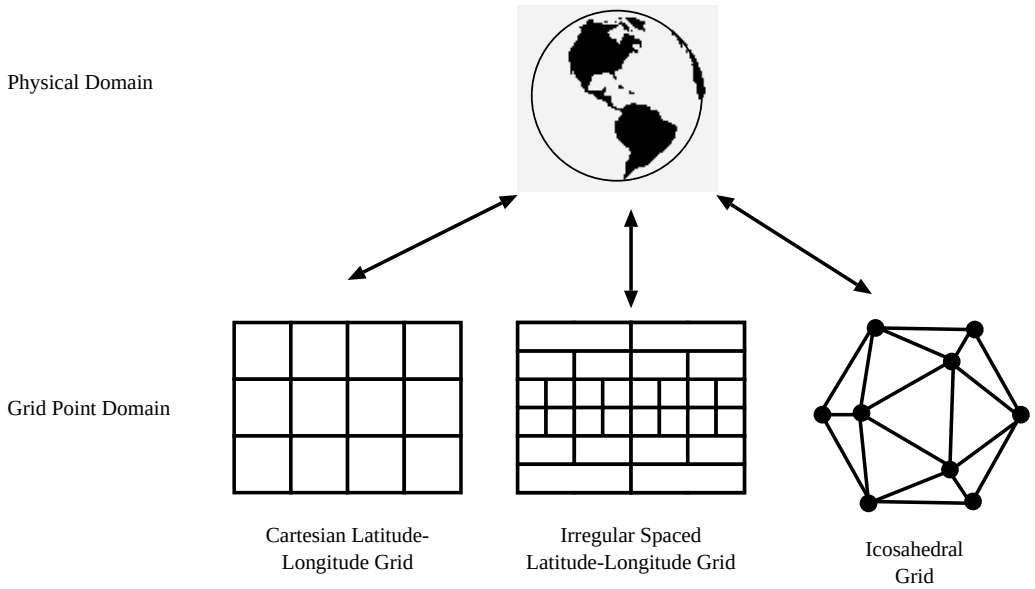

Figure 4.1: *Mappings from the physical domain to the grid point domain*

At NASA, a regular latitude-longitude-pressure coordinate system is used, where the pressure corresponds to the height. Thus, it is especially valuable to find a good decomposition for this representation. Consequently, all model variables are defined on each location of the grid. They are regularly distributed in relationship to the Cartesian grid, but irregularly distributed in relationship to the physical domain, the earth. This leads to more gridpoints at the poles than at the equator.

Irregular Latitude-Longitude Coordinate system

To ease the problem with the different density of the spatial domain in relationship to the physical domain, an irregular coordinate system can be used. It has fewer points at the poles. An irregular coordinate system has been used for the definition of the minivolume based OI. Internally, it is regularly distributed in each of a number of latitude stripes (Figure 2.8). Nevertheless, the overall distribution of the minivolumes is irregular for the spatial domain and the physical domain.

Spherical Coordinate system

Since the physical domain of the earth is a sphere, it can be advantageous to keep certain calculations in polar coordinates. An example for this, is the calculation of the difference between two locations. The great circle formula in the Cartesian space introduces a number of trigonometric operations. This is not necessary in polar coordinates where the difference between two points can be derived while using multiplications and additions.

Icosahedral Coordinate System

A way to improve the spatial and physical representation is to map the atmospheric fields and the minivolumes into an icosahedral grid[60, 8, 129]. Here, the distance between grid points is almost preserved[13]. Minivolumes and model variables are distributed regularly over the physical domain. In the traditional sense (Fortran90D/HPF), this distribution has to be regarded as irregular because the domain is represented by the nearest neighbor graph. Graph partitioning algorithms can be employed to find appropriate mappings for the grid points to the processors[113, 125, 112, 114, 106].

The icosahedral grid has the advantage of significantly reducing the storage space for the atmospheric fields. For a 2×2.5 grid, 91×145 points are maintained resulting in 13,195 grid points.

An icosahedral grid is constructed from an icosahedron (20 faces and 12 vertices). A grid is obtained by dividing the edges of the icosahedral into equal lengths and create new smaller equilateral triangles in the plane, and then project on the sphere[8, 60]. Different schemes are possible, which result in the same number of points on the sphere. There are $2 + 10n^2$ nodes and $20n^2$ faces in the k^{th} refinement of the triangulation, where $n = 2^k$. Table 4.1 shows the properties of the refinements, where h is the maximum arclength of any edge in the triangulation. The arclength representing 2 degrees is approximately 0.035 and for 2.5 degrees it is approximately 0.043. They are close to the the icosahedral grid with 10,242 points.

Hence, while using the icosahedral grid representation, a 22% reduction in the number of model variables can be achieved. This does not only reduce the volume for storing the model variables, but does also reduce the number of computations necessary.

Table 4.1: Dyadic icosahedral spherical triangulations

k	n	Nodes	Faces	h
0	1	12	20	1.107
1	2	42	80	0.628
2	4	162	320	0.326
3	8	642	1,280	0.165
4	16	2,562	5,120	0.083
5	32	10,242	20,480	0.041
6	64	40,062	81,920	0.021

For a Cartesian grid with 20 variables, 0.283 MWords are used for storing one variable of a field. During the calculation, the data assimilation is carried out over the past ten years. Hence, an atmospheric field can be stored in:

$$10 \times 365 days \times 4 \frac{1}{day} \times 0.283 MWords = 4.09 GWords$$

Using the icosahedral grid, only 3.19 GWords are necessary for storage. Currently, new algorithms are under development using icosahedral grids as a basis representation for climate models and assimilation systems[90].

4.4.2 Coordinate Systems based on the Minivolume Distribution

Previously, it was pointed out that the minivolumes are irregularly distributed over the sphere. Using a coordinate system based on the location of the minivolumes can simplify the domain decomposition of the calculations associated with a minivolume. Reducing the spatial domain to minivolume based coordinate system, reduces the number of calculations necessary. Nevertheless, the forward integration of the model will lead to inaccuracies[60, 75].

4.4.3 Coordinate Systems based on the Observational Domain Distribution

The observational domain is significantly different from the model variables and the grid domain. As seen in Figures 2.9-2.11, the observational data is clearly irregularly distributed

over the latitude-longitude grid. For the quality control algorithm, a regular decompositions can be found while mapping an approximately equal number of observations on each processor. It would be useful, if the observations close to each other are mapped onto the same processor, since a search in the vicinity of a point is conducted. Thus, it reduces the amount of data stored on each processor. The same applies to the OI algorithm. Different coordinate systems can be helpful for the determination of the distance between two coordinates.

4.5 Functional Decomposition

Two levels of functional decomposition have to be considered. First, the sequential program is divided into the three modules SLP, MIX and HUV analysis. Each module consists of a quality control step and the application of an optimal interpolation algorithm.
Second, the modules for quality control and optimal interpolation iterate over different domains. In correspondence with the data domain analysis, the loops are iterated over the observations (quality control) and the mini volumes (optimal interpolation). Each location in the domain is associated with a calculation.

4.5.1 Loop parallelization

As mentioned before, the data domains on which the main loops of the quality control and the matrix solve work are different. While the iteration for the quality control is pursued over the irregularly distributed observation data, the matrix solve is performed over the model variables in small minivolumes.
The data dependencies in the main loops for quality control and optimal interpolation, enable one to perform each iteration independently. Though, this kind of calculation is often referred to as *embarrassingly* parallel, it should be noted that due to the irregularities of the domain, the decomposition is not *embarrassingly* simple.
Thus, the original abstract formulation of the SLP, MIX and HUV algorithms (Programs 2.3-2.5) can be quite easily modified to reflect a functional decomposition. The new formulation adds a constraint of *processor locality* for

- each observation, in the case of the quality control,
- each minivolume, in the case of the matrix solve,

to the calculations attached within the loops (Programs 4.1-4.3).
To avoid redundant calculations, each calculation at the location of a minivolume/observation is assigned to only one processor. The resulting mapping of minivolumes/observations to processors is referred to as *data domain decomposition*.

4.5.2 Computational Irregularities

On each location, a calculation is performed. The time spent for the calculation depends on multiple factors. Its duration is difficult to predict. Thus, even if a regular representation of the domain can be found, load imbalance is introduced due to the different time needed to perform the calculation on each location.
The result of a calculation and the time to obtain the result depend on:

1. the location in the atmosphere,
2. the number the observations in the vicinity, and
3. the values of the observations.

Due to the extensive number of input parameters (in table form), an exact (quantitative) derivation of a performance prediction function is impractical. Nevertheless, a qualitative performance analysis, as shown in Section 4.7, can be obtained.

4.6 Data Domain Decompositions

Many different data domain decompositions are possible. Only the most useful decomposition, in regards to data and load balance, are considered.
The different domain decompositions, used in actual implementations of the parallel optimal interpolation and quality control algorithm, are classified in Figure 4.2. Distinguished are regular versus irregular decompositions, and dynamic versus static decompositions. Static decompositions are known before runtime, while dynamic decompositions are generated at runtime.
In spite of the fact that the locations of the minivolumes are not known at compile time, because they are stored in an external file. Static and dynamic decompositions can be dis-

Program 4.1 The parallel gross check algorithm.

```
1  proc Parallel Gross Check
2        T_h ← Tolerance
3        foreach observation [∈ processor] do
4          if the variable is not in allowed range
5            then mark observation as invalid
6          end if
7          if observation is (still) valid
8            then δ ← variable^O − variable^F
9                 if δ² > T_h((σ^O)² + (σ^F)²)
10                  then fail, mark observation as suspect
11                end if
12         end if
13       end foreach
14       [Update the validity on all processors]
15 end proc
```

Program 4.2 The parallel buddy check algorithm.

```
1  proc Buddy Check
2        foreach valid observation [∈ processor] do
3          Search for valid observations in a particular
4            radius around the data point.
5          analyzed value ← Perform univariate successive correction
6            method at the location of the observation with the help of S
7          δ ← observation − analyzed value
8          if δ is in tolerance level, δ ≤ ||T_h||
9            then mark observation as valid
10         end if
11       end foreach
12       [Update the reacceptance on each processor]
13 end proc
```

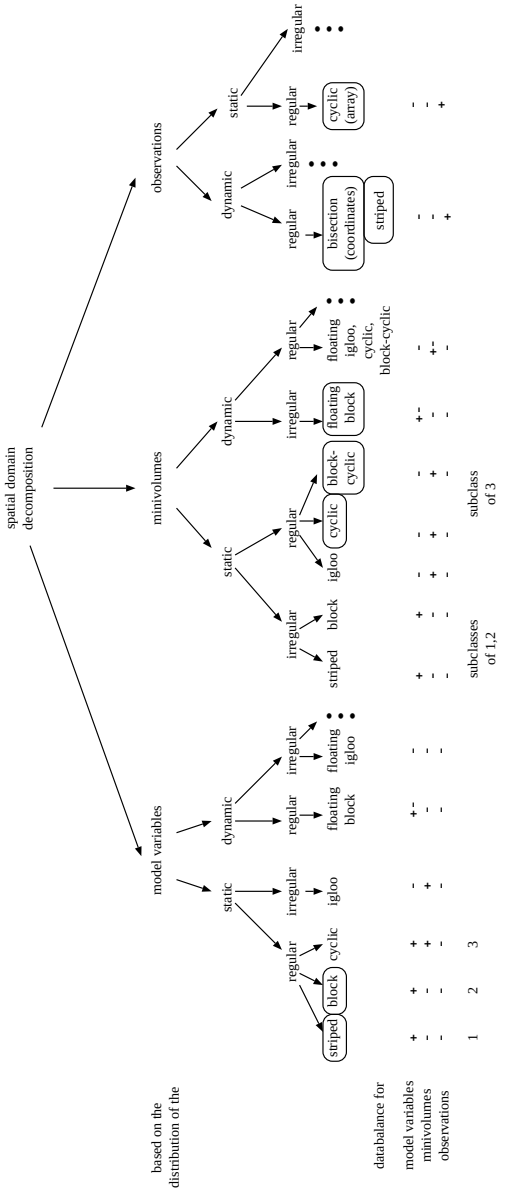

Figure 4.2: *Classification of the spatial coordinate mapping for the optimal interpolation*

Program 4.3 The parallel optimal interpolation algorithm.

```
1  proc Parallel Optimal Interpolation
2       foreach minivolume m [∈ processor]
3           Collect the data in the vicinity of this minivolume
4         foreach Vertical level
5             Calculate covariance matrix A_m
6           foreach grid point ∈ current minivolume
7               Solve the linear equation A_m x = b_g
8               Store the result
9           end foreach
10        end foreach
11      end foreach
12      [Forward the stored result to the processor writing the data]
13 end proc
```

tinguished for the minivolume based decompositions. This is because the minivolumes are static during all calculations.

First, decompositions based on the distribution of the model variables are explained. Model variables are specified in the Cartesian Coordinate system. The straight forward approach is to divide the regular latitude-longitude coordinate system in stripes or blocks. Each processor is assigned a block and performs the calculation on its block of data.

Figure 4.3 shows sample block decompositions of the model variables. The sphere (a) divides the earth into stripes of equal latitude width. The spheres (b) and (c) introduce longitude bands, in addition to the latitude bands. Because the block decomposition (c) does not preserve equal areas on the latitude-longitude grid, an example is given where a better physical decomposition is achieved. This decomposition is shown in sphere(d), and is referred to by atmospheric scientists as *igloo* decomposition. In the igloo decomposition, less latitude-longitude blocks exist towards the pole than, e.g., at the equator. The model variables are irregularly distributed with this decomposition.

Similar to the model variables, the minivolumes can be decomposed. In the extreme, where a minivolume is defined on each of the gridpoints, it is equivalent to the model variable decomposition.

Decompositions based on the geographical distribution of the observations are all dynamical.

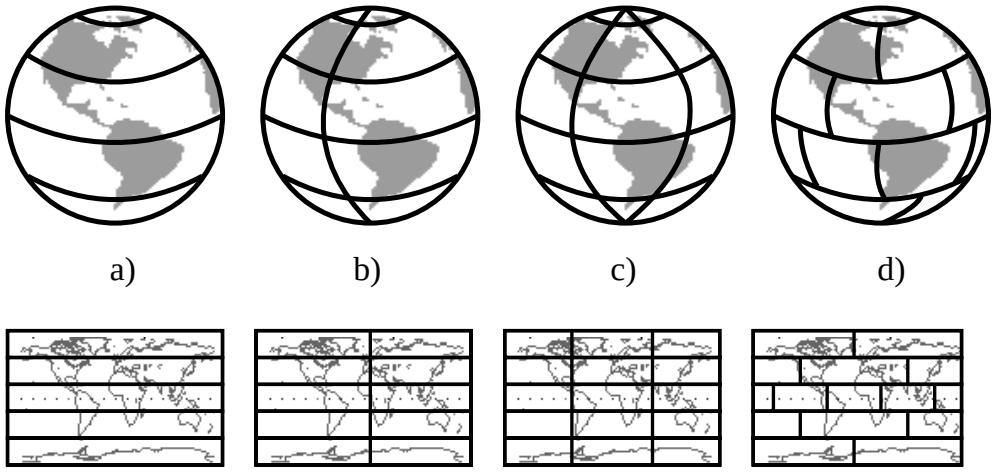

Figure 4.3: *Possible data distributions. (a) striped distribution, (b) +(c) blocked distribution, (d) irregular "igloo" distribution*

Dynamic decomposition strategies are explained in more detail later.
A static cyclic or block decomposition based on the position of the observation in memory is possible, instead of using the geographical location of the observations.
To evaluate the decomposition strategies, Figure 4.2 shows the quality of the databalance obtained by the different decompositions for the different domains. A "+" indicates good databalance, "-"' bad databalance, and "+-" not as bad databalance.
Of special interest are the decompositions emphasized by a frame. Even though experiments with most other decompositions have been performed, their results are omitted, because

- they achieve poor load balance,
- or are more complex than other equally well load balanced schemes.

4.6.1 A Generalized Specification for Decompositions

To allow experimentation with several domain decompositions, the original program has been extended in such a way, that a decomposition strategy can be used as input parameter to the program. The actual program does contain constraints of the form $\in$ *processor*, as used in the high level program description. This is done with the help of a set of functions specified

for each decomposition strategy d. The first function is a location function λ_d. It maps points from the coordinate space into the processor space:

$$\lambda_d(x, y) \mapsto p \in Processors$$

where p is a processor in the set of all processors, and (x, y) is a location in the latitude-longitude coordinate system. In case three dimensional decompositions are considered, an additional height(pressure) parameter can be specified:

$$\lambda_d(x, y, h) \mapsto p \in Processors$$

Since two different domains are used for the quality control and the optimal interpolation, two different location functions are possible. This can be distinguished by an additional index to the function, indicating the domain on which the function is defined:

$$\lambda_d^O(x, y, h) \mapsto p \in Processors \quad \text{and}$$

$$\lambda_d^G(x, y, h) \mapsto p \in Processors \quad ,$$

where O indicates the observation domain, and G, the grid or minivolume domain.
The second function returns the set of all points in a processor, determined by the domain decomposition. To distinguish between the observation and the minivolume decomposition, the function ω_d^O is used for the observations, and the function ω_d^G is used for the minivolumes, respectively:

$$\omega_d^O(p) \mapsto \text{ set of observations assigned to processor } p,$$

$$\omega_d^G(p) \mapsto \text{ set of grid points assigned to processor } p,$$

Since a calculation can depend on observations located in other processors, the definition of an *overlap* is useful. The overlap $\gamma_d^O(p)$ is the set of all observations which are needed for the calculation, but not assigned to the processor p. In the case of the NASA assimilation system, all observations which are 1600km of any observation in $\omega_d^O(p)$, but are not included

in $\omega_d^O(p)$ itself, are included in the overlap. It should be noted that an observation can be in overlaps of different processors. For obtaining the first guess values, an overlap region on the grid domain exists. The overlap is defined by $\gamma_d^G(p)$.
Besides the specification of these functions, a set of redistribution routines are provided, which automatically extract and redistribute the required observations onto the different processors. For future implementations, more sophisticated redistribution libraries for this task can be found in[94, 16].

4.6.2 Overlap region

Now, that a better understanding of the distribution is provided, a closer look at one of the processors will explain the relationship between the regions assigned to different processors (Figure 4.4). Each processor is responsible for all calculations assigned to the minivolume/observation locations, which are embedded in its assigned geographical region. This region is referred to as the *interior region*. To do so, each processor must have access to

1. the model variables embedded in the region,
2. the model variables in the overlap,
3. all observational data in the region, but also to the observations in an overlap region with other processors.

The size of the overlap region is determined by the *region of influence* of the observation points. The physical model, as described in Chapter 2, uses only observations in a finite vicinity of an observation. Global knowledge about all points is not necessary. The radius of influence is chosen to be be 1600 km due to its good tradeoff between the quality of the solution and the limited number of points participating in the calculation. The observations in the overlap region are all observations which are in 1600km distance from the observation contained in the region, but are not assigned to the interior region.
Two cases occur for a minivolume/observation:

1. All necessary observations are embedded in the interior region assigned to the processor.
2. Some observations are located in neighboring processors, but are in 1600km of a point in the interior region.

The first case does not involve a change in the original sequential algorithm. For the second case, the observations contained in the overlap regions have to be gathered before the calculation is started. Gathering all observations, of all overlap regions between the different processors, before the actual calculation is performed, allows one to perform the calculations of the main loops independent from each other. No data exchange is necessary, since all data dependencies are resolved[116].

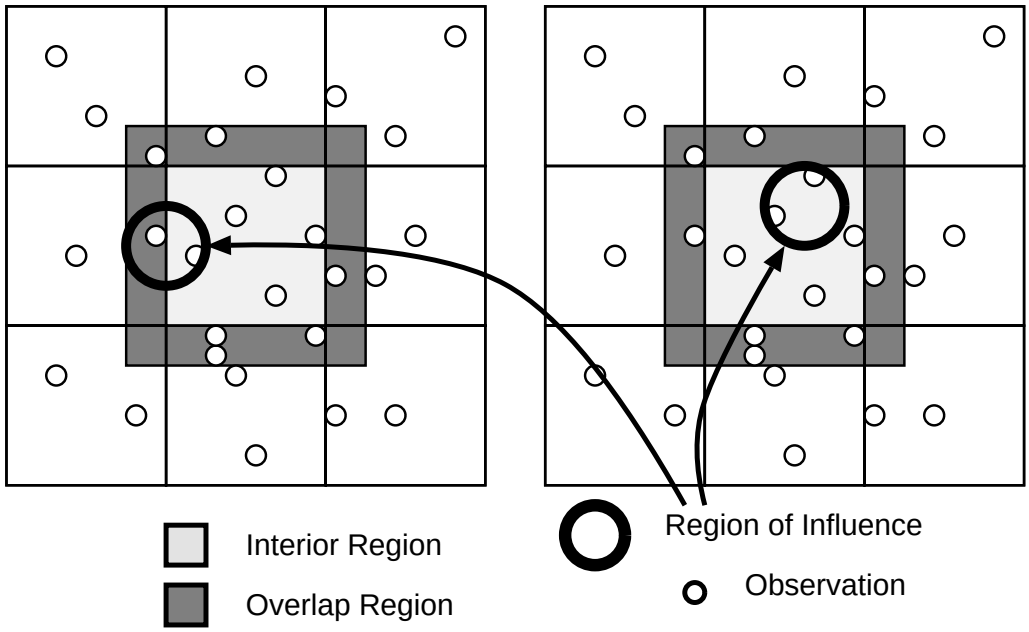

Figure 4.4: *The definition of an overlap region.*

To achieve high efficiency, the number of elements in the overlap region should not be too large in order to keep the ratio between the time spent for the calculation and the data exchange as large as possible.

Many algorithms in literature (e.g. finite element methods) have regular sized overlap regions. This is useful for the derivation of fast algorithms to gather all data in an overlap region. In the case of the Parallel Optimal Interpolation (POI), a simple Cartesian based range search algorithm[96] can not be applied to determine the observations contained in the irregular overlap regions. Figure 4.5 gives an example for the overlap region of a processor containing the observation data over Europe. For low latitudes, at the equator, the search region is a

perfect circle, while it is distorted for high latitudes. On the poles, the search region has a rectangular form, in regards to the grid domain. It contains all observations up to the latitude, which is 1600km away from the pole.

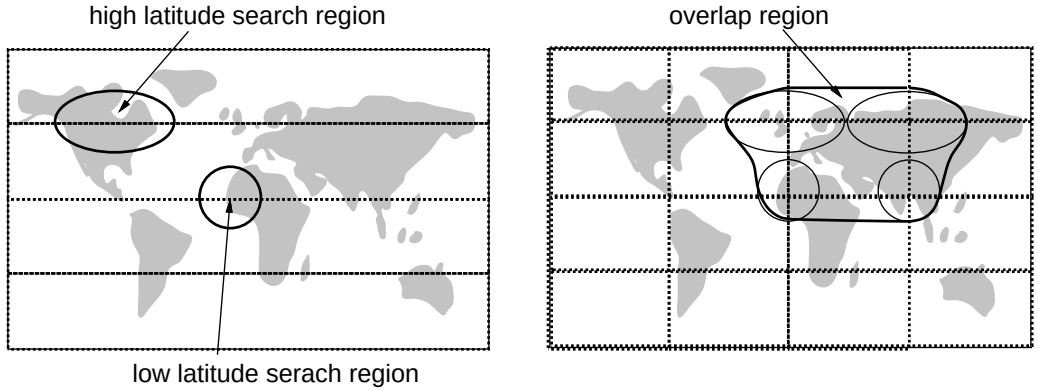

Figure 4.5: *Irregularities in the overlap region.*

The number of observations contained in the overlap region is only known at runtime. The search for data elements in the overlap region can be determined via a two step process. Assume that the observations are sorted in longitudinal direction. Assume that the coordinates are available, both in Cartesian form, and in the spherical coordinate system. The observations in the overlap are obtained by the following two stage process:

1. Discard all observations which are 1600km away in longitudinal direction. This can be done with a simple comparison.

2. For the rest of the elements, test if they are 1600km away from a point in the interior region.

The first step is performed to avoid the second, more expensive, calculation.
While regular overlap regions are well understood in computer science [47, 116], irregular overlap regions are cause for active research[66].

4.6.3 Memory Considerations

The different data distributions allow one (theoretically) to reduce the amount of data to be stored in the different processors. The original serial algorithm assumes that all variables in

the different domains are accessible.

For the parallel OI(POI) algorithms, different classes are distinguished according to the storage of the model variable and the observation data. Then, the following *storage pattern* classes exist[121]:

POI(global grid, global observation): All model variables and observations are stored in each processor.

POI(global grid, local observation): All model variables are stored in all the processors but each processor stores only the observational data which are necessary for the calculation.

POI(local grid, local observation): All processors store only the model variables and observations which are necessary for the calculation.

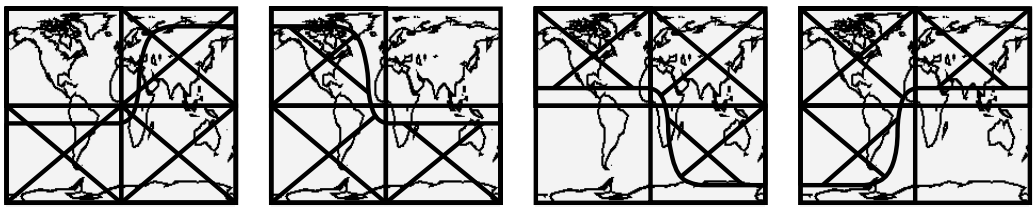

Figure 4.6: *In each processor, the entire model variables and only the necessary observations are stored.*

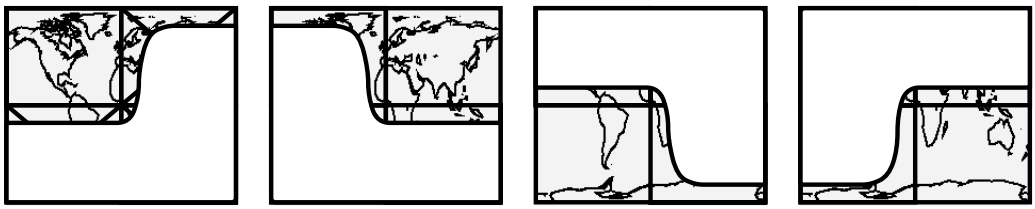

Figure 4.7: *In each processor, only the necessary model variables and observations are stored.*

Storage Pattern (Global Grid, Global Observation)

A straightforward way for parallelization is to store all the data available in each processor (Figure 4.6). This scheme is well known as data replication. Each processor can work inde-

pendently on its assigned sub domain, once data dependencies are resolved. Load balancing can be implemented without the need of restoring data.
The large amount of memory used in each processor and the high IO bandwidth for initialization are among the disadvantages. A lot of data will not be used during the calculation, causing unnecessary overhead.
Fortunately, there is enough memory available in current supercomputers to support this strategy for the desired problem instances. In addition, it is expected that the production version for the next few years will not have more then 150,000 observations. Preprocessing in a separate program will ensure this [28].

Storage Pattern (Global Grid, Local Observation)

To reduce the memory usage in each processor, only observations necessary for the calculation are stored. Due to the smaller number of observations stored in a processor, the algorithm used to determine observations in the vicinity of a point is significantly faster.
The decision whether the observation is necessary, is performed with a special detection algorithm. In certain domain decompositions, we can speed up the detection algorithm, while presorting the data, with the help of a parallel recursive bisectioning algorithm[117, 116]. This will help to improve the load balance between the different processors. Program 4.4 shows how to incorporate the sorting algorithm.

Program 4.4 The presort algorithm.

```
1 proc  Read observations
2       Each processor reads a number of observations in parallel
3       Presort the observations in parallel
4       Redistribute the observations depended
5         on the domain decomposition
6 end proc
```

Storage Pattern (Local Grid, Local Observation)

The best algorithm, in terms of the memory requirements, is to store only the grid data and observations necessary for the calculation in each processor (Figure 4.7). Because of the

irregularities of the domains, it is quite a difficult task to find a good domain decomposition. Unfortunately, the design of the existing program did not allow us to explore this strategy. A complete redesign of the original sequential code is necessary to implement this strategy. This was the original project approach. The development was stopped on initiative of NASA due to the realization, that it was impossible task by only one programmer to do the sequential and parallel program development. Nevertheless, it is worthwhile to consider this method in future research because the model (GCM) can be implemented in this fashion. Therefore, a sophisticated dynamical load balancing strategy is a good alternative.

4.7 Load Imbalance

More information about the actual runtime behavior of the OI algorithm is provided in Figure 4.8. Here, for each minivolume column the actual runtime of the loop is depicted. The darker the area around the minivolume, the more time was needed to complete the calculation.
The average CPU time for a calculation is 0.539s, while the standard deviation is 0.164s. The minimum and maximum times are 0.22s and 0.81s. The times at the poles are 7.39 and 8.05 seconds. This introduces quite a big load imbalance, even if the poles are considered separately. In case each minivolume-column would be calculated on a separate processor, the machine can only be utilized with approximately 66 % efficiency. Hence, it is desirable to map an appropriate number of minivolume columns on a processor to increase the efficiency. In case the pole calculations are included, the maximal efficiency possible is approximately 14%.
A better load balance can be achieved by using either irregular static decompositions or dynamic load imbalance strategies.
The differences in the load distribution of the quality control depends on the number of observations in the vicinity of the actual calculated observation.

4.7.1 Dynamic Load Balancing

In this part, we describe a dynamic load balancing strategy for the OI algorithm. Let, a task consist of the generation of the covariance matrices for a column of minivolumes and all matrix solves associated with this minivolume column. The straight forward approach is

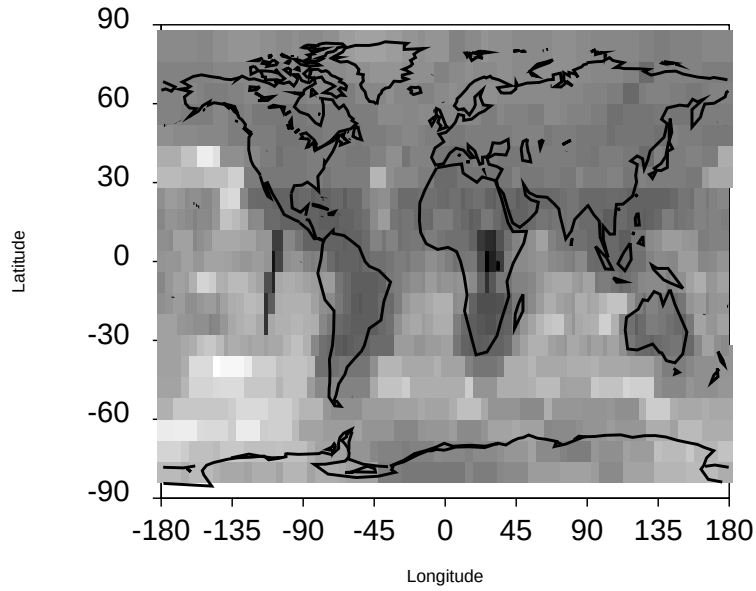

Figure 4.8: *Load imbalance caused by the calculations performed at the different profiles.*

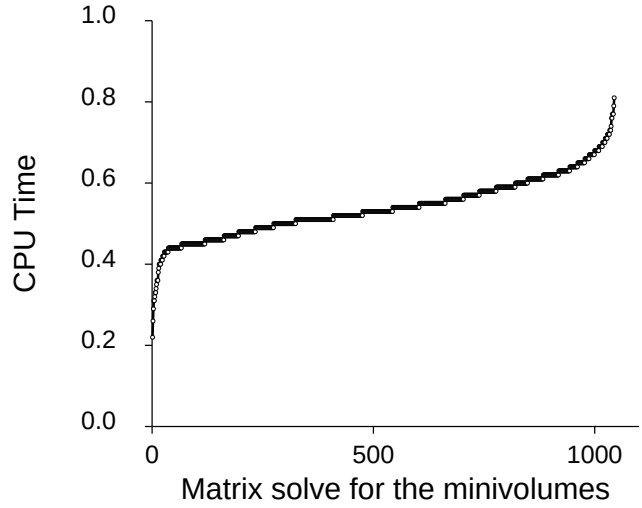

Figure 4.9: *Timings for the calculation of the statements for the set of minivolume at a particular location on the globe. The times are sorted by their value.*

to design a dynamical load balance algorithm with central control (Program 4.5). A host monitors the list of tasks. The calculations are performed in a set of slave processors. Once a slave has finished its work, it sends the result to the host. Then, the host takes the next task to be calculated from the task list and submits the work to the slave without work. This is done, as long as, all tasks have been completed.

If the duration for calculating a task is small, in comparison to the time needed for the submission of the task, then the previous algorithm will not perform efficiently. Some of the processors can not be utilized because the host can not serve the requests in time. To avoid this situation, the tasks consist of *columns of minivolumes* and not a single minivolume.

A more general scheme can be derived by submitting a number of tasks instead of only a single task. This avoids the expensive start-up time, existent in most MIMD machines, while sending large messages instead of multiple smaller ones.

Another improvement can be made by using multiple hosts or a decentralized control. Program 4.6 shows an algorithm where each node includes the function of a host. Thus, the control is distributed over the slaves. The actual host processor is only needed to gather the results at the end of the calculation.

The domain is split into many small, but sufficiently large blocks. Once a processor is finished with the calculation of its blocks, it looks for a processor which still has tasks to do. The requesting processor interrupts the other processor and the task will be submitted, so that a better utilization of the processors is achieved.

In practical use, floating blocks build the tasks as depicted in Figure 4.10. Here, six processors are used to solve a problem which is decomposed in 24 blocks. The numbers in each block specify at which time step a task is started. When a task is finished, a new task is started at the current time. In the example, the processors 2 and 3 have finished their work at the beginning of time step 5. The processors 1 and 4 have more work than they can handle in time step 5, thus, the tasks are redirected to the processors 2 and 3.

To make this algorithm efficient, the cost to migrate a task to another processor should be small. This is the case when data replication is used and the message consists solely of a number assigned to each subregion. If more than one task is left in the region, the work is equally split in half.

Similar strategies based on igloo or other decompositions can be derived. Many possible algorithms and strategies for dynamical task scheduling can be found in literature[2, 33].

Program 4.5 The program for a global controlled dynamical load balance algorithm.

```
1  proc Host
2        Start up all slaves
3        Decompose the domain in tasks t_1, ..., t_n
4        Assign to each processor a task
5        while Still tasks there do
6              t ← Next task from the task list
7              p ← Wait for a jobless slave
8              r ← Receive Result from p
9              Send the task t to p
10       end
11       Terminate Nodes
12       Write result
13 end proc
14
15 proc Slave
16       while ¬terminate do
17             t ← Receive Task from Host
18             r ← Calculate task t
19             Send to the host that the slave is jobless
20             Send the result r to the host
21       end
22 end proc
```

Program 4.6 The program for the floating task load balance algorithm.

```
1  proc Host
2        Start up all slaves
3        Decompose the domain in tasks t_1, ..., t_n
4        Assign to each processor a number of tasks t_i(p), ..., t_j(p)
5        foreach p ∈ Processors do
6          Receive results from p
7        end
8        Terminate Nodes
9        Write result
10 end proc
12 proc Slave
14       thread thread_1
15             while ¬terminate do
16                   foreach t ∈ Local tasks
17                     Remove t from the task queue
18                     r ← Calculate task t
19                     store Result r
20                   end
21                   foreach (p ∈ Neighboring Processors ∧ work received) do
22                     Send request for work to neighboring slave p
23                     (t_a, ..., t_b) ← Receive Local tasks
24                     work received ← (t_a, ..., t_b) ≠ null
25                     Include t_a, ..., t_b in the local task list
26                   end
27                   if there is no more work from the neighbors
28                     then terminate ← true end if
29             end
30             Send the results to the host
31       end thread
33       thread thread_2
34             while ¬terminate do
35                   if (another slave q requests work)
36                     then if (still tasks there)
37                               then send a number of tasks to the slave requesting work
38                                else send null to q end if
39                   end if
40             end
41       end thread
42 end proc
```

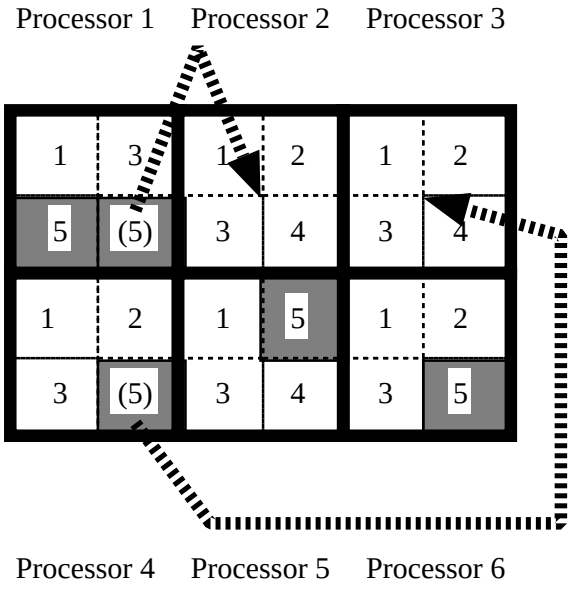

Figure 4.10: *Dynamical loadbalance strategy with the help of floating tasks.*

4.7.2 Geographical Static Decompositions

For many applications the load imbalance can be reduced with the help of cleverly chosen irregular static decompositions. While analyzing the load distribution of Figure 4.8, one immediately realizes that a correlation exists between the number of observations, the outline of the continents and the time to pursue the calculation on a minivolume column. One could implement a strategy which finds a balance between:

- the less time consuming calculations over the oceans,
- the less time consuming calculations where no data points are available,
- and the more time consuming calculations over the data rich continents.

The problem with a static geographical decomposition is that the observation data (and the calculation time for each gridpoint) change hourly, daily and seasonally due to different observation distributions and values. Static decompositions based on geographical information can only be an approximation to a good decomposition.

4.7.3 Data Balanced Decompositions

Since the correlation between data and load changes for each analysis, a good static load balanced decomposition can only be estimated. Thus, the term *data balanced* decomposition is more appropriate for classification than using the term load balanced decomposition. In other words, the data balance does not guarantee load balance.
Even though data balanced decompositions are more difficult to obtain for the model variables and model grid of the OI, they are easy to achieve for the observation domain of the quality control. In the quality control, there exists a strong correlation between the number of observations and the execution time in the processor. Thus, instead of choosing a geographically striped or block decomposition, one can chose a data balanced decomposition.

4.7.4 Cyclic Decomposition

Using a scattered decomposition can resolve the problem of load imbalance in many cases.
A special case is the *cyclic* decomposition. As the name indicates, all necessary calculations are distributed in a cyclic fashion over the processors, in order to avoid geographical regularities and irregularities of the domain. This is done under the assumption that the cyclic decomposition spends, on average, the same amount of time on its computational tasks, thus little or no load imbalance will occur.
This strategy can be employed, equally well, for the quality control and the OI. The only exceptions are the calculations over the poles, which take longer time for the matrix to be solved.
Thus, a two stage decomposition for a parallel OI is chosen. First, the domain is split into three blocks. Two of the areas consist of the poles, while the rest of the physical domain is placed in its own area. Each area is distributed in a cyclic fashion over the processors. While the domain over the poles are cyclicly distributed for each vertical level, the rest of the area is distributed in cyclic fashion for each minivolume column. Figure 4.11 depicts this distribution. The distribution is referred to as *block-cyclic* distribution. Thus, the final decomposition is built upon:

1. an irregular block decomposition, and

2. a cyclic decomposition of each block, in either horizontal or vertical direction.

Communication between the blocks is unnecessary because the problem is transferred to an embarrassingly parallel program with the help of data replication.
Furthermore, this decomposition is the basis for providing a load balanced HPF program. As mentioned before, the disadvantage of this algorithm is the high memory overhead due to data replication. Data replication is necessary to achieve high efficiency, while reducing the number of messages exchanged. Fortunately, the sizes of the problem instances considered today, fit well in the memory of a single node on state of the art supercomputers, and also on heterogeneous workstation clusters.

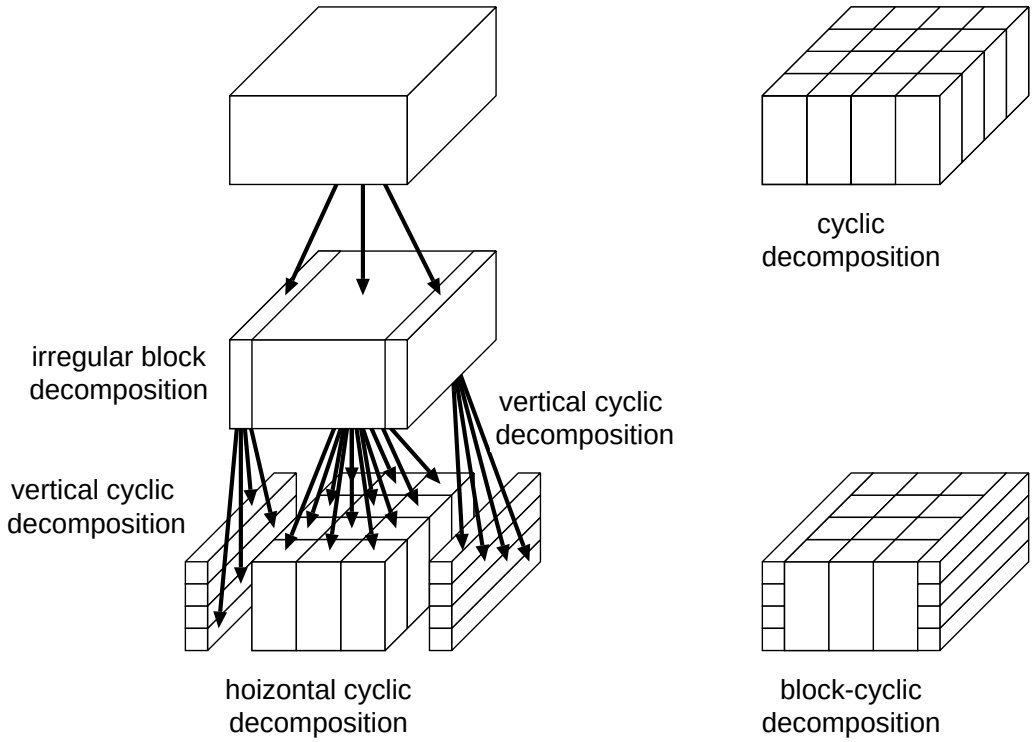

Figure 4.11: *Block-cyclic distribution*

4.7.5 Random Scattered Decomposition

For the cyclic decomposition, the number of processors has to be chosen carefully. The number of points in the horizontal and the vertical direction on the grid should not be a

multiple of the number of processors. In this case, the algorithm will degrade to a simple striped decomposition.
While using a *random scattered* decomposition, this problem is avoided, too.
In a random scattered decomposition, each location is assigned randomly to a processor. This can be done by a random permutation on an arbitrary decomposition, which partitions the domain in approximately equal parts.

4.8 Evaluation of the Data Domain Decomposition Schemes

A good decomposition of the domains is essential for the definition of an efficient parallel algorithm. Different data decompositions for the quality control and the OI algorithm have to be considered. Dynamic load balance enables one to provide an algorithm suited for both. A static cyclic decomposition, with special treatment at the poles, provides a scalable algorithm. The cyclic decomposition has the advantage that it provides a way to formulate the algorithm in both,

1. message passing and
2. data parallel (HPF)

programming paradigms. Using both paradigms provides maximal portability on many machines. The disadvantage of high memory consumption is eased under the realistic assumption that enough memory is available on a processing node.

4.9 Future and Related Research

4.9.1 Modifying the Functional Decomposition

Because the SLP and MIX data assimilation algorithms do not scale well with very high numbers of processors, it would be desirable to include as much functional parallelism in the code as possible. The original assimilation system is performed in three steps, as shown in Figure 4.12 (a).
What would happen if the three steps, e.g. SLP-Analysis, MIX-Analysis, and HUV-Analysis, are executed in parallel and not in sequential order?

The answer is obvious: It would enable a much better utilization of a parallel machine with an even higher number of processors. One argument that supports the existence of such an algorithm, is that in nature the physical processes happen in parallel and not in sequential order. The PSAS algorithm[57] does not distinguish between the in three different modules anymore.

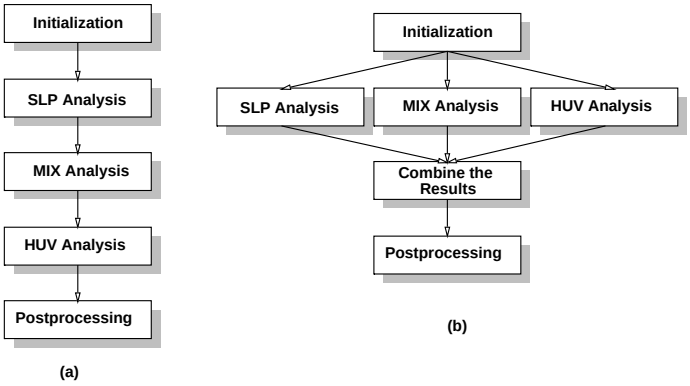

Figure 4.12: *High level flowchart of the optimal interpolation algorithm. (a) original algorithm (b) suggested change for a parallel algorithm.*

4.9.2 Loop Restructuring

Taking into account that the data dependency between observations is only 2-5mb below and under each horizontal layer, it is clear that a vertical decomposition is a natural extension of the horizontal decomposition. It has the advantage of being much more fine grained than the parallelization over a set of minivolumes, in the same profile. The extension to the parallel program is depicted in Program 4.7.

Another important consequence of the use of parallel computers is the increased computational power available. Thus, it is possible to eliminate the minivolume concept and perform the optimal interpolation on each grid point with increased accuracy.

Program 4.7 The parallel OI algorithm looping over minivolumes and vertical levels.

```
1  proc Parallel Optimal Interpolation
2        foreach Vertical level [∈ processor] do
3          foreach minivolume m [∈ processor] do
4              Collect the data in the vicinity of this minivolume
5              Calculate covariance matrix A_m
6            foreach gridpoint g [∈ minivolume m] do
7                Solve the linear equation A_m x = b_g
8                Store the result
9            end foreach
10         end foreach
11       end foreach
12       [Forward the stored result to the processor writing the data]
13 end proc
```

4.9.3 Dataparallel Assimilation Systems

HPF

The algorithm for MIMD machines can be converted into an HPF algorithm, since the parallelization is done over the loops. The necessary data exchange is performed before and after the loops are executed.

In case the model variables are stored in all processors, the only addition to the program would be to distribute the minivolumes and the model variables in an appropriate way.

If a cyclic decomposition is used, all observations should be stored on the processors. With the use of the block decomposition, the observations can also be distributed in a block fashioned way.

Nevertheless, one would have to obtain the dimensions of the region covered by the decomposition for the model variables or minivolumes, in order to determine all observations in the overlap. These observations should be stored in a separate list. The interior of the algorithm has to be modified for the search procedure which obtains all observations in a vicinity of a location in the atmosphere. This irregularity is not supported while using the current HPF standard. Nevertheless, INDEPENDENT loops provide enough semantical and syntactical features, to formulate the algorithm in HPF.

The amount of work for rewriting the program in this way, requires far less work than rewriting the the program for a parallel MIMD algorithm.
While using the cyclic decomposition, a representation of the algorithm is found which can be implemented on SIMD, as well as, MIMD machines. The internal representation of an HPF program would be analog to its message passing version.
Unfortunately, at the time of writing, the compilers were not stable enough to consider this solution[58]. Stable compilers will be available soon. Extensions for irregularly distributed domains will be especially useful, to describe more complex domain decompositions. Even the implementation of an icosahedral distribution would be possible.
An experimental implementation of the quality control algorithm can be found in [81].

Vectorized Optimal Interpolation Algorithm

The parallel algorithm for vector supercomputers[103, 65] is analog to the parallel algorithms introduced for the MIMD architectures. The vectorization takes place on the loops over the minivolumes and observations, respectively.
Under the assumption that the observations are randomly distributed in memory, the vectorized program relates to the cyclic decomposition for the MIMD algorithm. In case the minivolume list is reordered randomly in memory, the cyclic decomposition corresponds to the vectorized quality control.
The current vectorized algorithm does not assume that the observational data, nor the minivolumes, are in a particular order. If the observation data is sorted by latitudes, the load imbalance is influenced on newer models of vector processors. Newer vector supercomputers are built using a number of processors. Each processor is responsible to calculate part of the vector. In the case of independent loops, the vector can be mapped in chunks onto each processor. The calculation can be performed analog to the MIMD striped bisection decomposition for the quality control, and the striped decomposition for the minivolume OI. Because of the different times needed to perform a calculation at a location, load imbalance is introduced and the efficiency of the algorithm is reduced. One has to realize that data locality, while accessing observations and model variables from the memory, is essential for the efficient usage of data caches.
Comparisons between the order of the problem domain and the data cache access have not been conducted, since MIMD algorithms are in the foreground of this research.

A striped decomposition is especially of importance, due to the future availability of a Cray C90, with 16 nodes, at NASA GSFC. Using a few number of processors based on a data balanced decomposition for the quality control, can lead to superlinear speedup, as shown in Section 6.3.2.

4.9.4 The ECMWF Box Distribution

At the European Center for medium Weather forecast, a different optimal interpolation algorithm has been previously parallelized for MIMD computers[67]. The motivating factor for a domain decomposition is the unevenly distributed data. As described before, the availability of the data varies from six hour to six hour period, and from day to day.
The method developed at ECMWF uses a *box* method. For each analysis step, the globe is partitioned dynamically into about 1500 boxes, dependent on the density of the observation data. The first box size is about 600×600 km, but uses the data of a box of about 1000×1000 km. In case there are more observations included in a box than is given by a special threshold, the box will be divided into four smaller boxes. This process will be iterated, as long as, the number of observations in its box is not smaller than the threshold value.
An example of a partition obtained with the box algorithm is shown, in Figure 4.13. The closeup in Figure 4.14, shows the iterative splitting mechanism. Here, a threshold of 451 observations is used. The domain can have up to two vertical levels, if there are more than 451 observations together in the upper or lower level.
The control of the distribution is performed by a global host supervising a farm of processors. The host controls the work distribution to the node processors. Each node asks for work and gets an assignment from the host (calculate a particular box with the help of its observation data).
The algorithm was originally implemented on a Cray YMP, with 8 processors. Intercommunication between processors has been achieved via files.
The box algorithm is similar in approach to the bisectioning algorithm, as introduced in Section 4.9.4. The difference is that the bisection algorithm tries to keep the amount of data better balanced for each node. Thus, a bisection based quality control algorithm will have a better performance.
Incorporating the box algorithm on the NASA OI code would have the advantage of generating boxes of predefined maximal size. Big grid blocks do not exist, as they can appear in the

bisection algorithm at regions of sparse observations. Thus, in the case of the OI, the box algorithm will have a better load balance under the assumption that the observation data distribution is sparse, as in the SLP and MIX analysis. For the HUV analysis, the data is dense in most regions, and the algorithms will have a similar performance.
The biggest disadvantage of the box algorithm is its central control. Congestion can occur when there are too many nodes asking for work than the master can handle. The machine would be strongly underutilized. To prevent congestion, multiple solutions are possible:

1. The calculations at each node should be large enough, such that, the fraction between calculation time and communication time is as large as possible. Nevertheless, it can occur that multiple processors are finished at the same time and have to wait in line for their next job assignment.

2. The central roll of only one host can be relaxed, while partitioning the set of nodes and providing each set with its own host. This can be done in a hierarchical manner, in order to reduce communication between hosts[2] (see Section4.7.1).

3. The elimination of the host processor and the generation of a completely decentralized version of the OI (see Section4.7.1).

The decentralization can be achieved by an initial static mapping, as introduced earlier, and by nearest neighbor communication of the workload between the nodes itself. Since the nearest neighbor communication introduces further complications (e.g., a node might have no work, but its neighbors are still busy), a static strategy is often preferable, in case the load balance problem is small.

4.9.5 Alternatives to OI

Currently, many alternatives to the OI algorithm are under development. It includes methods based on descent methods and Kalman-Bucy filters. The most similar algorithm to the OI is the Physical space System Assimilation System (PSAS). In contrast to the OI, PSAS sets up the entire covariance matrix for all observation correlations and solves the large matrix at once. No distinctions between the SLP, MIX and HUV analysis are necessary. Due to the size of the matrix, the system can only be solved with state of the art supercomputers. The

experience gained from the OI algorithm and its parallelization has a large influence on the development of the PSAS algorithm. As in some domain decompositions for the POI, PSAS uses a bisectioning algorithm to distribute the observations equally onto different processors. Superobbing, like in N-body calculations, are used to reduce the data dependencies between the observations, thus reducing the matrix to be solved. FORTRAN90 has been established as programming language to express PSAS. More strict software methods have been employed.

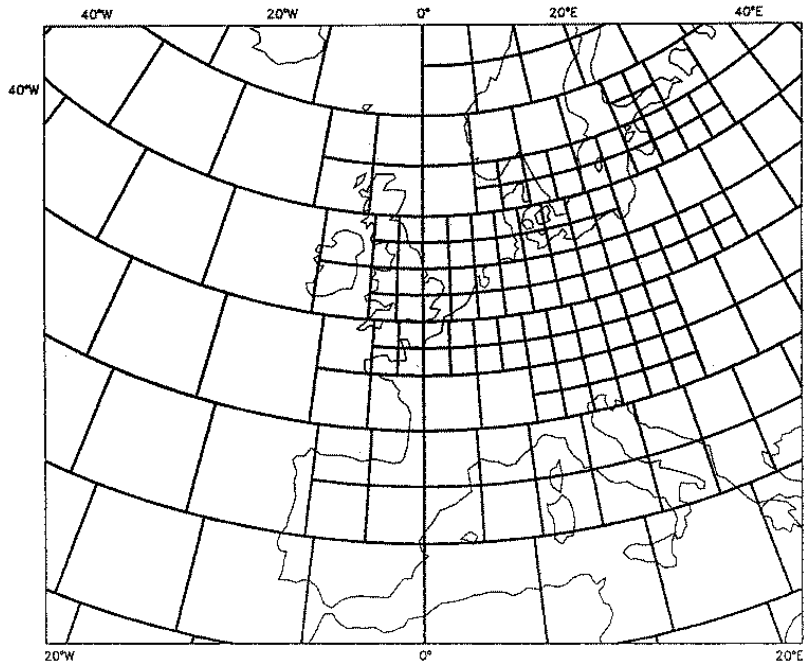

Figure 4.13: *An example of a decomposition obtained by the ECMWF box decomposition scheme.*

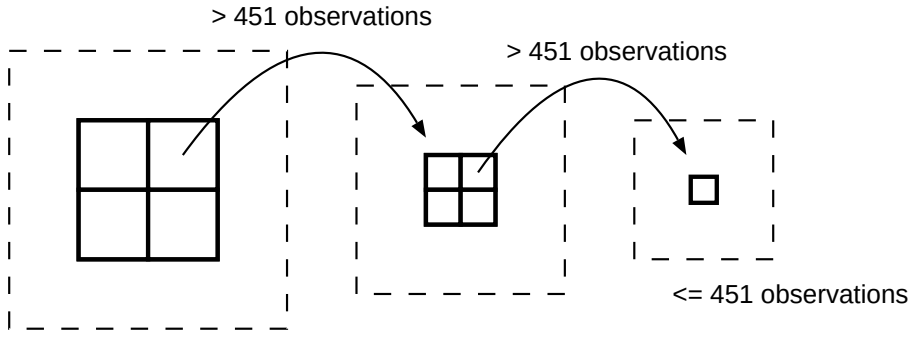

Figure 4.14: *The hierarchical decomposition of the ECMWF box decomposition scheme. The process is iterated for each box, as long as it contains more than 451 observations.*

Chapter 5

Deterministic Quality Control

One of the findings of the research pursued, is the proof that the quality control, as provided in the original data assimilation system, is not as good as previously thought. A new algorithm could be derived and was introduced at [123]. The decision has been made to modify the existing quality control algorithm and provide a better version to be implemented in PSAS at DAO.

The reason why this problem has not been discovered before, can be based on a lack of documentation of the program and the complex structure of the source code. The requirement of obtaining the exact numerical results between the parallel and the sequential code led to the discovery. Unfortunately, at the time of implementing the parallel algorithm, the correctness of the sequential algorithm was assumed by the originator team. Providing a new algorithm which even produced a different result was suspicious and a mistake in the parallelization of the algorithm was concluded. Thus, it was necessary to take the original sequential algorithm and rewrite it in such a way, that input and output is handled in the same way, like the parallel algorithm. The comparison between this algorithm could provide the same numerical results as the sequential algorithm. The differences between the parallel algorithm were slight, but they were existent. A further modification which emulates the parallel algorithm on the sequential machine, showed finally the differences. After an extensive analysis of the differences and the incremental debugging of the code, the error was located in the quality control.

The solution of the quality control was dependent on the order of the observations

in the input data.

The explanation for the invariance can be seen in the Programs 2.3 and 2.4 for the gross check and buddy check. Once an observation fails or passes the quality control, this property is marked in a global accessible array. This is done in the loops of the buddy and gross check. Since the quality depends on the observations in the surrounding area, a data dependency has been created between the instantiations of the statements included in the buddy-check loop. One way to solve this problem is to postpone the rejection/reacceptance of the observations until the loops are completely iterated.

This can be done with a simple data structure emulating a set. In the case of the grosscheck, all rejected observations are included in the set of rejected observations.

In the second step, the buddy check loops over the set of rejected observations and tests if they should be included in the set of reaccepted observations.

As shown in Programs 5.1 and 5.2, the sets are updated after the corresponding loop has been iterated completely.

5.1 Physical Interpretation of the Quality Control Problem

The question arises about whether the misbehavior of the quality control algorithm should be ignored because the differences are quite small in the test case we considered. In addition, it should be found out if the problem magnifies, while parallelizing the sequential algorithm. Figure 5.1 shows a physical explanation of the problem, in a real coordinate space. The first two rows represent a quality control cycle for the buddy check. They are iterated in different order. The last row shows how a parallel algorithm can effect the first case.

Assume that there are two observations 1, and 2, which are accepted before the quality control starts. A number of observations (3-11) are obtained in consecutive order, in close distance to the observations. Let, the observations 3 to 11 be referred to as the *observation line*. All observations in the observation line are rejected, but should be included in the calculation.[1] Each of the observations is in the vicinity of at least a couple of previous observations. The observations 1 and 2 influence only the first couple of observations in the observation line.

[1] It can be a storm which has been not detected beforehand (last 3-6h).

Program 5.1 The modified gross check algorithm.

```
1  proc Gross Check
2        T_h ← Tolerance
3        terminate ← false
4        while (¬terminate) do
5              foreach valid observation ∈ processor do
6                 if variable is not in allowed range
7                    then mark observation as invalid
8                 end if
9                 if observation is (still) valid
10                   then δ ← variable^O − variable^F
11                        if δ² > T_h((σ^O)² + (σ^F)²)
12                           then fail, mark observation as suspect
13                        end if
14                end if
15             end foreach
16             Update the validity on all processors
17             terminate ← Are no observations rejected in the loop?
18       end do
19 end proc
```

Assume, the quality control iterates from observations 1-11. Then, a cascading effect occurs, resulting eventually in the acceptance of the necessary observations (see first row of Figure 5.1).

In case the loop is iterated in reverse order, only the observations closest to the observations 1 and 2 are accepted.

In case this algorithm is parallelized and the observations are distributed on different processors, as depicted in the last row, the situation becomes even more problematic. The cascading effect is stopped at the processor boundaries and the effect becomes worse.

A way to avoid this problem is to introduce an iterative quality control algorithm. During each iteration of the loop, rejected observations are included in the set of rejected points. The rejection is marked only after the loop is completed. This process will be iterated as long as the set of rejected observations is not empty. For a parallel algorithm, it is necessary to include a communication step between the loops, forwarding the new results to processors

Program 5.2 The modified buddy check algorithm.

```
1  proc Buddy Check
2       terminate ← false
3       while (¬terminate) do
4            foreach suspicious observation ∈ this processor do
5              Gather all observations in the area around the suspicious observation
6              analyzed value ← Perform univariate successive correction
7                method at the location of the observation with the help of S
8              δ ← observation − analyzed value
9              if δ is in tolerance level, δ ≤ ||T_h||
10               then just mark observation as reaccept
11             end if
12           end foreach
13           Include all reaccepted observations in the set of valid observations
14           Forward the result to the other processors.
15           terminate ← Are no observations rejected in the loop?
16      end do
17 end proc
```

which have overlap regions (regions of vicinity between two observations). In the practical application, only 1-5 percent of the observations are expected to fail the gross check. Thus, the communication step is limited and can be performed quite quickly. The analog strategy can be applied to the buddy check.
This iterative quality control algorithm has two important properties:

1. It is specified in such a way that the global structure between the sequential and the parallel algorithm can be maintained.

2. The parallel algorithm will produce the same result as the new sequential algorithm.

5.1.1 Optimization in the Report Generation

Another important improvement is done in the algorithm which reports the results for the analysis by the atmospheric scientist. This has been done before in a routine of complexity $O(n^2)$, where n is the number of observations. This routine has been improved while using

better data structures. The complexity of the new algorithm is $O(n)$. The additional space required is $O(1)$.

5.2 Future Research

Most quality control failures take place in groups. To reduce the number of observations, recognizing the fact that most of the statistical variances occur in geographical clusters, techniques from pattern recognition can be employed to detect clusters of observations. The speed of the quality control can then be increased, while performing separate quality control instantiations on a cluster of observations, instead of the whole data set.

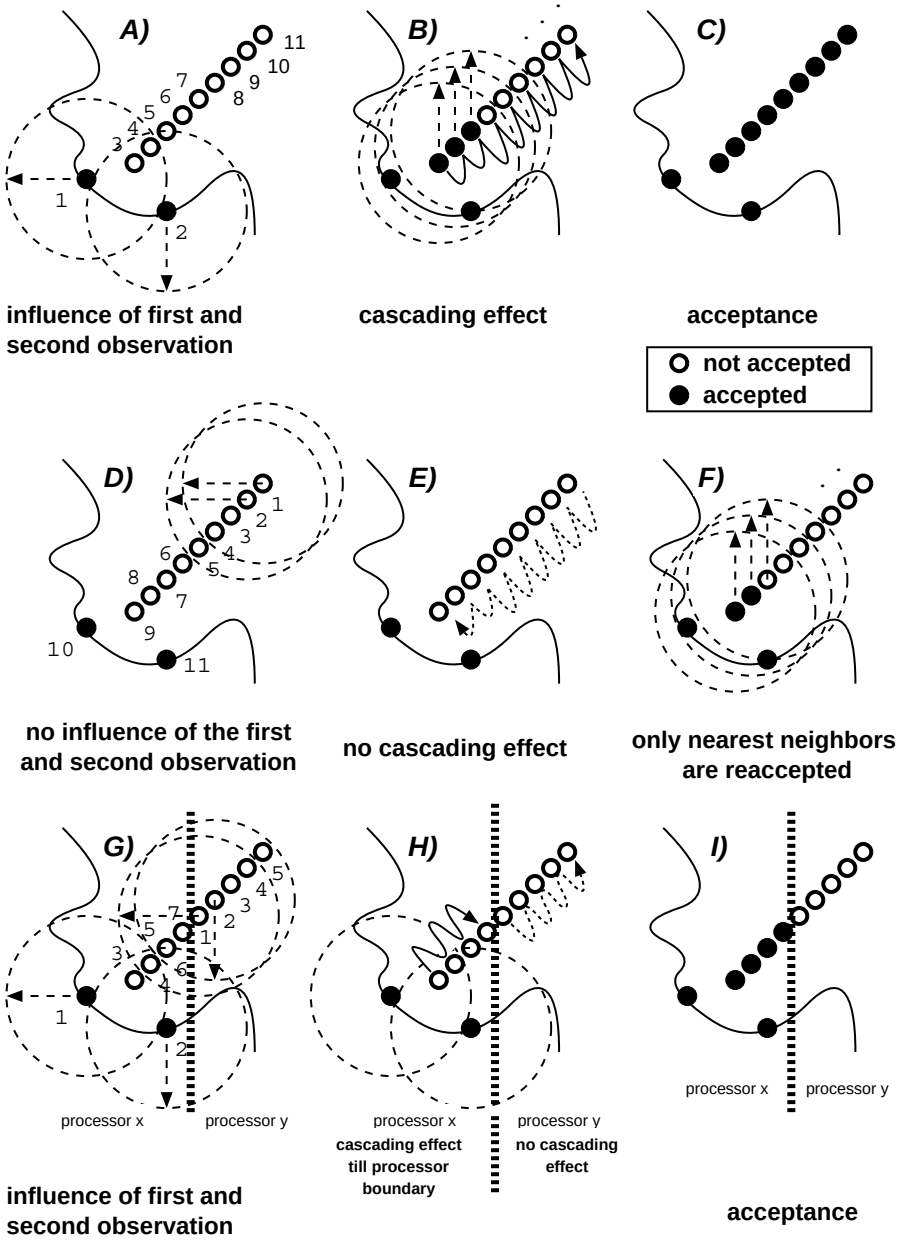

Figure 5.1: *Illustration of the quality control problem. The first two rows show examples for the acceptance of observations for different orders of the observation in the input. The last row shows the acceptance when two processors are used, while traversing the observations in the same order like the first row.*

Chapter 6

Experimental Results

In this chapter, the experimental results obtained with the sequential and parallel programs for the assimilation systems are presented.
Several different versions of the sequential assimilation system have been released by the NASA Data Assimilation office. The experiments reported here are based on two different versions of the code. They are known under the version numbers 1.2s and 2.0mv[84, 85]. We refer to them from now on as Version 1.2 and Version 2.0.
Version 1.2 is a slightly modified version from the original data assimilation system, as used in the production at DAO. This version has recently been superseded by Version 2.0. The experiments for each version are listed separately in this chapter. A comparison between their performance is given.
Findings obtained during the parallelization of Version 1.2, have a direct impact on the parallelization strategy used in Version 2.0 of the assimilation system. This reduces the software engineering effort drastically. First, some characteristics of the hardware platforms as used for the experiments, are given.

6.1 Hardware Used for the Experiments

Different hardware platforms were used to pursue the code development and the program performance measurement. This includes a Cray C90, a vector-supercomputer, the SP2 MIMD parallel computer from IBM, and a DEC Alpha workstation farm, with a fast network connection.

The DEC Alpha Workstation Farm at Northeast Parallel Architectures Center

The DEC Alpha Workstation farm at NPAC consists of 8 DEC 3000/400 systems, each with 62 MB of memory, running DEC OSF/1 v2.0. Each system has a FDDI interface. The FDDI interfaces connect the systems over fiber optic cable to a DEC Gigaswitch. Six of the workstations have 467MB swap space, while the two remaining ones have 342MB and 280MB swap. The size for the instruction and data cache is 8KByte each. The performance is enhanced by a secondary unified cache of 512Kbytes. The worksations operate in time-sharing mode.

The SP2 at Northeast Parallel Architectures Center

The SP2 at Northeast Parallel Architectures Center has 12 nodes, from which 8 are *thin* nodes and 4 are *wide* nodes. The access to the wide nodes is restricted. The other 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 currently contains 256 Mbytes. An external disk of 1.0GB is available for each node. The nodes of the SP2 operate in time-sharing mode.

The DEC 3000/400 Alphas and the RS6000 are rated individually, as depicted in Table 6.1[32].

Table 6.1: Some Performance characteristics of a single Alpha and SP2 node.

	DEC Alpha		RS6000	
Model	3000/400		390	
SPECint92	74.7		114.3	
SPECfp92	112.5		205.3	
LINPACK 1000x1000	90	MFlops/s	181.0	MFlops/s
LINPACK 100x100	26	MFlops/s	53.0	MFlops/s

The SP2 at Cornell Theory Center

The SP2 at the Cornell Theory Center (CTC), is the largest available IBM RS/6000 Scalable POWERparallel System (SP). The system has 512 processors and its peak performance is advertised with over 130 gigaflops. Of the 512 processors, 48 are *wide nodes*. The memory capacities of the wide nodes range from 256 megabytes to 2 gigabytes. Of the nodes, 464 are *thin nodes* with 128 or 256 megabytes of memory each. The Andrew File System(AFS) is used to provide uniform access to the nodes. Each node has a local disk space. For large numbers of processors, it is necessary to replicate the input data on the local disks in order to avoid congestion to external fileservers. If not done properly, it will lead to a failure of AFS. All 512 nodes have an aggregate of 80 gigabytes of memory (RAM) and 1.2 terabytes of local (internal) disk space.
A mass storage in the form of a UniTree is available, employing high-speed transfers directly between a network (HiPPI)-attached RAID disk cache and a HiPPI-equipped SP2 node. The capacity of this mass storage system, measured in terabytes, is virtually unlimited. The machine uses Easy-LL as batch operating system.

The SP2 at Maui High Performance Computing Center

The SP2 at Maui High Performance Computing Center has 400 nodes, with 53 wide nodes, of which 5 have 1Gbyte memory and the rest have 256 MByte. 248 of the nodes are thin nodes, with either 64 or 128 MByte memory. 26 Nodes of the SP2 are used for interactive work. The rest of the nodes are reserved. Like in Cornell, a UniTree provides mass storage access.
The machine uses LoadLeveler as batch operating system.

The Cray C90 at NASA Goddard Space Flight Center

The Cray C90 at Goddard Space Flight Center, has 6 CPUs with a peak performance of 1 GFlops per CPU. It has 256 MWords (here, a word is 64 bit) central memory. Two Very HIgh Speed (VHISP) channels are connected to a Solid-State Storage Device (SSD) containing 512 megawords (here a word is 64 bit). Five mass storage systems which can hold 4.8TByte of data, are attached to the Cray. After the experiments were completed, the system was upgraded to a J932, a J90 with 32 processors. The machine uses NQS as batch operating

system.

6.2 The Dataset Used for the Performance Analysis

Both versions of the data assimilation system are applied on problem instances provided by the DAO for the program verification. Two datasets have been generated.
The first dataset was provided in ASCII form and has been converted into the appropriate binary format for the machines used. The underlaying coordinate system is a 4 by 5 degree latitude-longitude grid with 14 height levels, resulting in a domain of $72 \times 46 \times 18$ grid points. 1046 Minivolume columns are distinguished. About 15,000 minivolumes divide the grid points in small subregions. The total amount of observations is approximately 100,000. This dataset was used for the first experiments with the assimilation system of version 1.2s. This dataset is referred to as *dataset A*.
With delivery of Version 2.0, a second dataset with 14 vertical levels has been generated in IEEE format. This dataset uses a coordinate system of 4 by 5 degrees. 1046 minivolumes are used. This dataset is referred to as *dataset B.* The distribution of the observations have been displayed in previous chapters (Figures 2.10, 2.9, and 2.11). It reflects the observations taken on January 22, 1989 at the synoptic time 12:00. The distributions of the observations are not significantly different form each other, since they are taken at the same synoptic time during the same month.

6.3 Experiments Based on Version 1.2 of the Assimilation System

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 spent 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 latter, are of utmost importance because accounts on state of the art systems with up to 512 nodes are available to the project.
Since the algorithmic concepts of the SLP and the MIX analysis are similar to the HUV analysis, and the sealevel and moisture analysis use only 12% of the total computation time,

the emphasis has been to parallelize the HUV analysis. The parallel algorithms for SLP and MIX look the same.

Nevertheless, since the number of observations and minivolumes is so much smaller (the SLP analysis is performed only on the surface), the algorithms are less efficient for large numbers of processors. For the MIX analysis, the sequential algorithm is used because the parallel algorithm performs actually slower, due to the increased message passing overhead.

In the rest of the chapter, results are presented for the more time consuming HUV analysis.

6.3.1 Performance on a RS6000

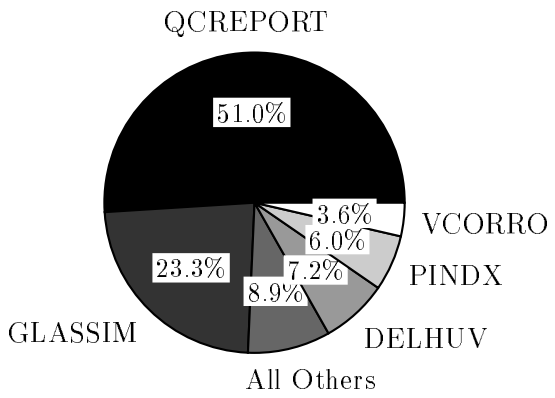

Figure 6.1: *Fractions of the computational intense parts of the sequential assimilation program, Version 1.2.*

Figure 6.1 shows the performance distribution of the most time consuming parts of the Version 1.2, based on dataset A. The routines depicted have the following semantics:

glassim is the main driver routine which includes many more subroutines than depicted in Program 2.1[84].

qcreport performs statistical operations during the quality control on the dataset and model variables and prints the result.

vcorro performs the vertical interpolation of the observation errors.

pindx is an index function used to access observations, in the list of observations.

delhuv performs the HUV optimal interpolation without the quality control.

all other routines, including the MIX and SLP analysis and the HUV quality control, have much smaller runtimes.

It was surprising to find out, that the optimal interpolation algorithm did not consume most of the CPU time, but a report generating routine which is an integral part of the quality control algorithm. Following Amdahls Law, it was important to optimize this routine. Amdahl's law states [47, 4]:

> If an inherently sequential component of a problem takes a fraction α of the time on a single node, then one can never achieve a speedup factor greater than $\frac{1}{\alpha}$, no matter how many processors are used.

Since more than 50% of the time is spent in programs which were initially not subject to a parallelization, a speedup of 2 is maximal possible. Thus, it is clear that some effort had to be redirected to improve the sequential program performance.
The original algorithm for the data report and analysis uses $O(n^2)$ computations on each horizontal level to obtain statistical data, useful for atmospheric scientists to evaluate. While introducing more sophisticated data structures and reordering the loops of the original algorithm, an optimized version with complexity of $O(n)$ has been derived. The performance of the new quality control algorithm reduced the calculation time for typical datasets by over 50% on RISC machines.
The runtime for one instantiation of the optimal interpolation is 1200 seconds, while it is 2000 seconds for the quality control (including the report routines). This does not relate to the expected algorithmic performance of the quality control and the optimal interpolation. It is expected, that the time for the quality control algorithm is only a fraction of the time spent for the optimal interpolation.

By optimizing many parts of the sequential programs, using Basic Linear Algebra Subroutines(BLAS), removing unnecessary reporting routines, and invoking sophisticated compiler options during the compilation, the original time used for the HUV quality control and the optimal interpolation algorithm could be reduced by an additional 50%.
The results have been forwarded to the DAO and were used as supporting evidence, that a restructuring of the sequential algorithm is necessary.

6.3.2 Performance of the Parallel Algorithm

The first parallel algorithms, developed for the version 1.2, were based on striped spatial domain decomposition of the model variables, as well as, the data balanced domain decomposition of the observations. In each processor, only those observations are stored which are needed for the calculation.
A comparison of the latitude bands generated with the striped and data balanced decomposition is displayed in Figures 6.2-6.7 for 10,20 and 40 processors. It is obvious that fewer observations are located, e.g., on the poles in contrast to Europe. Therefore, the data balanced stripes on the poles are much larger than in Europe. Consequently, this leads to the problem that different numbers of minivolumes are assigned to the processors. Thus, load balance is introduced for the matrix solve.
One limitation of these parallel OI algorithms, is that the ratio between overlap region and actual computational region should be small, to provide a high degree of parallelism. For both decompositions, this is the case for small numbers of processors.
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 bisecting algorithm[117]. The bisection strategy distributes on each processor an approximately equal number of observations.
While the strategy performs well for the quality control, it performs less efficient for the optimal interpolation. This is to be expected, since the time of performing the calculations assigned to each processor depends on:

- the number of observations in the processor, in the case of the quality control, and
- the number of gridpoints, which are located in the interior region assigned to the processor, in the case of the optimal interpolation.

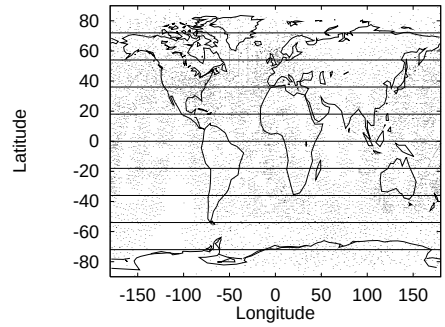

Figure 6.2: *Striped data decomposition onto 10 processors.*

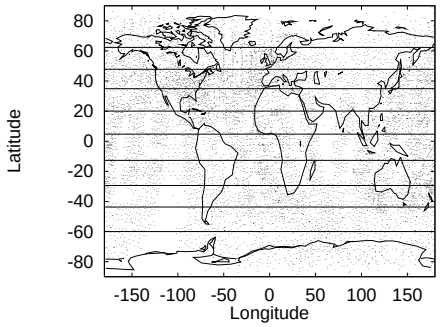

Figure 6.3: *Data balanced striped decomposition onto 10 processors.*

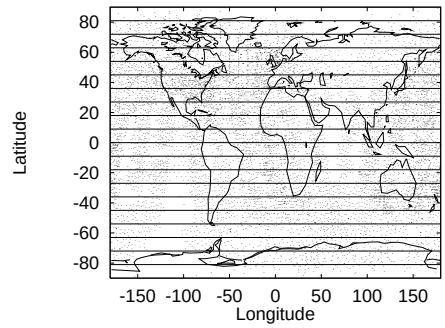

Figure 6.4: *Striped data decomposition onto 20 processors.*

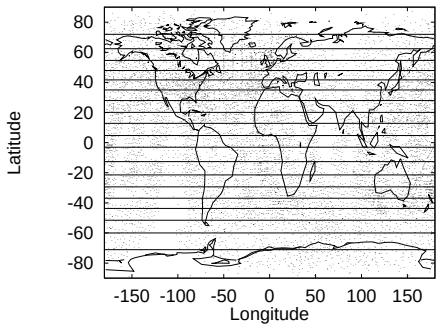

Figure 6.5: *Data balanced striped decomposition onto 20 processors.*

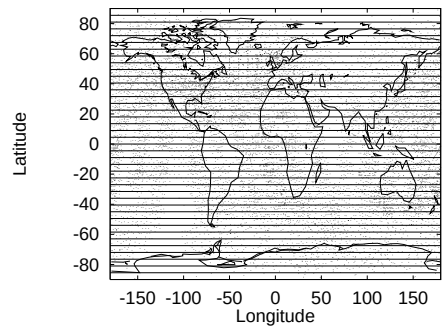

Figure 6.6: *Striped data decomposition onto 40 processors.*

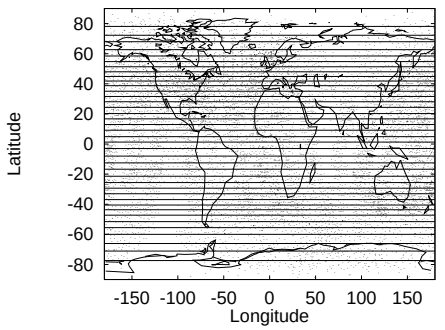

Figure 6.7: *Data balanced striped decomposition onto 40 processors.*

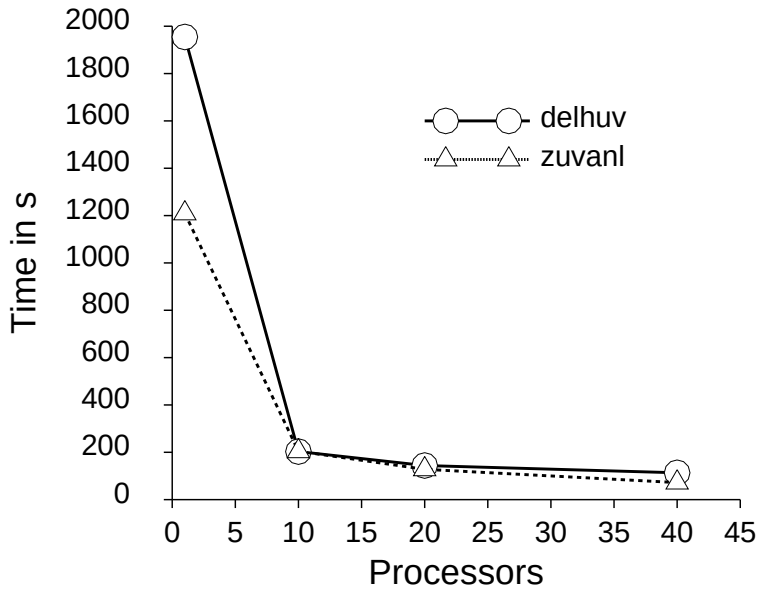

Figure 6.8: *The calculation time vs. the number of processors used for the HUV analysis, using the striped decomposition.*

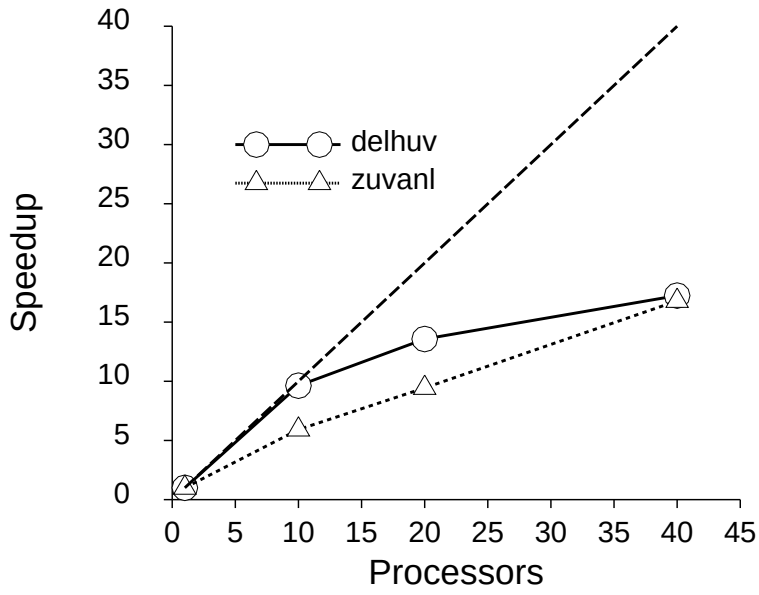

Figure 6.9: *The speedup vs. the number of processors used for the HUV analysis, using the striped decomposition.*

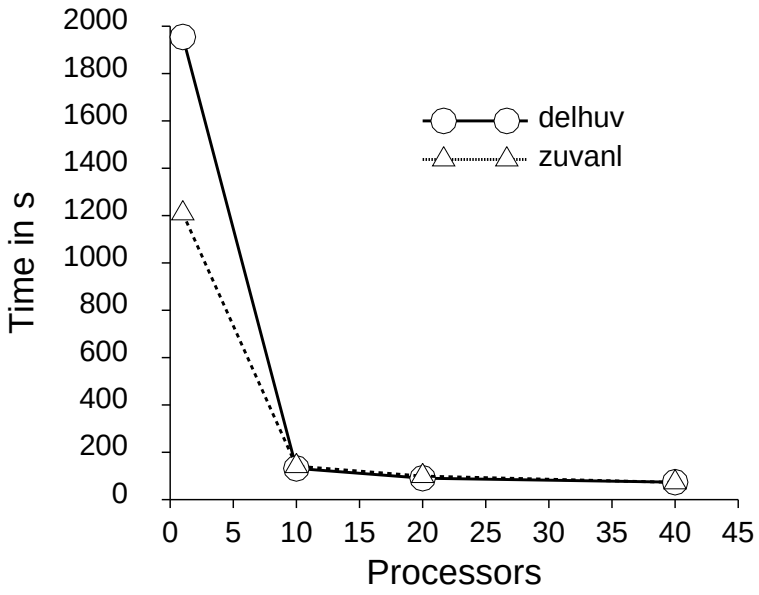

Figure 6.10: *The calculation time vs. the number of processors used for the HUV analysis, using the data balanced decomposition.*

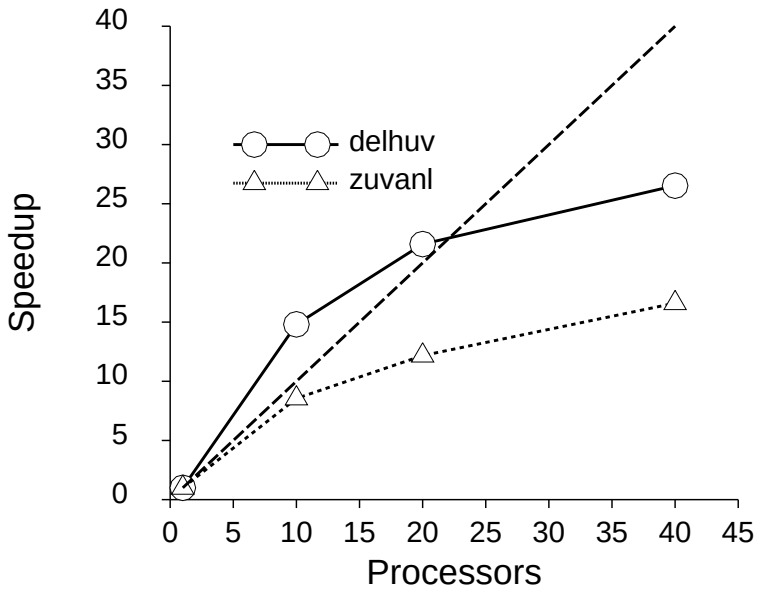

Figure 6.11: *The speedup vs. the number of processors used for the HUV analysis, using the data balanced decomposition.*

If large numbers of processors are used for the optimal interpolation, the time for setting up the covariance matrix of a minivolume will dominate the calculation, instead of the time to perform the Cholesky factorization.

The Figures 6.8-6.11 display the results for the most expensive computational parts of the data assimilation system. They are the HUV quality control, abbreviated with *delhuv*, and the HUV optimal interpolation algorithm, abbreviated with *zuvanl*[84, 85]. The timings and the speedup obtained are depicted for both distributions with different numbers of processors.

If small numbers of processors are used, a good speedup for both algorithms is achieved. This 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 superlinear speedup is obtained during the quality control. This is motivated by the following facts:

1. To determine the neighbors influencing an observation, the modified search routine is used. The more data points are stored in a processor, the larger is the performance gain while using the improved search algorithm. This is the case for large latitude bands, as obtained while using few processors.

2. While increasing the number of processors, the search of observations in the overlap region becomes more dominant and the speedup slows down.

The performance can be improved further, while introducing different domain decompositions. For the rest of the chapter, all other experiments are conducted with Version 2.0.

a block distribution, or even better, a cyclic distribution[121].

As expected, it is advantageous to use a domain decomposition which tries to achieve data balance. Thus, the observations should be evenly distributed for the quality control. For the optimal interpolation, the minivolumes should be equally distributed over the processors. The reason why the performance of the domain decompositions, introduced so far, perform less efficiently for higher numbers of processors, is analyzed next.

6.4 Experiments Based on Version 2.0 of the Assimilation System

Switching 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 version of BLAS. Another important change was to eliminate and rewrite certain diagnostic functions in the quality control. The qcreport routine has been disabled for the production runs. The dataset B is used for the experiments with the version 2.0mv (see Section 6.2).

6.4.1 Sequential Program Analysis and Performance

Cray Execution Summary

Previously, a detailed analysis of the performance of the assimilation system was not available. To compare differences between the code running on vector-supercomputers and MIMD-parallel supercomputers, using message passing, a time consuming performance analysis on the Cray C90 has been conducted. The performance data is obtained while using only one processor and switching on all hardware performance monitoring capabilities.

Table 6.2: Performance of the HUV analysis on a Cray C90.

CPU seconds	86.68	CP executing	20803075880
Million inst/sec (MIPS)	48.38	Instructions	4193422250
Avg. clock periods/inst	4.96		
% CP holding issue	66.90	CP holding issue	13916409486
Inst.buffer fetches/sec	0.65M	Inst.buf. fetches	56266068
Floating ops/sec	133.25M	F.P. ops	11550125021
Vector Floating ops/sec	131.12M	Vec F.P. ops	11365242632
CPU mem. references/sec	115.49M	actual refs	10010354957
avg conflict/ref	0.76	actual conflicts	7626901523
VEC mem. references/sec	108.70M	actual refs	9421725526
B/T mem. references/sec	2.99M	actual refs	258859362
I/O mem. references/sec	0.06M	actual refs	5265781
avg conflict/ref	0.51	actual conflicts	2707169

Table 6.2 depicts the performance data of the assimilation system, as gathered on a Cray C90. The performance data on the Cray is obtained with the help of the hardware performance monitoring (*hpm*) routine. A detailed description of the values shown in the table can be found in the technical manual[25]. Most of the timing numbers depicted in Table 6.2 will not be explained further. We leave those numbers in the table, in order to provide a complete overview. The important numbers are the number of floating point operations and instructions, as executed during the program run. The quality of the loop vectorization is specified by the Cray code analysis software as "medium".

Running the optimized assimilation system, modified for a RISC machine, leads to reduced performance on a Cray. The operational four dimensional data assimilation system (GOES 1.3) performs at 306MFlops. This includes the quality control, the optimal interpolation and the model forecast calculation. The objective analysis algorithm based on the optimal interpolation (Version 2.0) performs inbetween 165-186MFLOPS. The modified unoptimized algorithm as run on a single processor of an MIMD machine performs with 133MFlops on one processor of the Cray[64].

Performance of the Sequential Algorithm on RS6000 and Dec Alpha Workstations

Table 6.3 shows the performance of the HUV analysis and the HUV quality control, as well as, the routine reading in the observation data. All times are given in seconds. The runtime of the sequential algorithm compiled with different options on SP2 nodes and DEC Alpha workstations are displayed.

Table 6.3: Performance of the Sequential Program

	SP2	SP2	SP2	DEC Alpha	DEC Alpha[1,2]
Node type	thin	thin	wide		
Compilation options	-O2 -pipa	-O2	-O2	-O2	-O4
Read Observations	1.21	1.91	1.22	6.55	6.53
Quality Control	44.38	41.51	38.72	82.26	78.80
Optimal Interpolation	621.39	818.23	639.63	660.55	590.55
Total time	667.05	861.80	679.64	749.73	675.93

Note 0: all times are given in s
Note 1: while aligning common variables, no performance improvement is achieved.
Note 2: with -O4 -math math_library fast, not much improvement

More interesting are the times for the SP2 because of the availability of large the supercomputers based on the RS6000. Unfortunately, it is not possible to compile the OI code on an RS6000 with loop restructuring (-O3 and higher) without influencing the numerical stability of the computation. Inlining, performed with the option -pipa improves the runtime about 23%. Since this compiler optimization step takes a long time during the compilation, the timings given for the parallel code are obtained only with the optimization level O2 on thin nodes of an SP2. In case wide nodes should be available, the performance can be improved by another 22%. The increased performance does not influence the parallelization strategy of the assimilation system.
The fraction of the runtime of the quality control and the optimal interpolation between the RS6000 and the DEC Alpha is remarkable. While the quality control performs almost twice as fast on the RS6000, the optimal interpolation is only slightly better (while using optimization level 2). This can be explained by the fact that the quality control algorithm uses less data. Cache mismatches and swapping are less frequently. The cache and swap space in the Alphas is bigger than on the RS6000.

6.4.2 Performance Comparison of the Parallel Version 1.2 and 2.0

As depicted in Figures 6.12 and 6.13, the new assimilation system has different hot spots in contrast to the older version on the SP2. In the old version, the times for the HUV quality control and optimal interpolation were almost the same. In the new version the zuvanl constitutes to 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. The runtime for the optimal interpolation has been improved by a factor of about 2.
An overall comparison of the speedup between the Version 1.2 and 2.0 is presented in Table 6.4. Here, an SP2 with 40 nodes is used. Due to the limitations of the striped decomposition, a result of Version 1.2, further decomposition strategies were explored for Version 2.0. Nevertheless, the performance on 40 processors using the striped and the cyclic decomposition is almost the same for the optimal interpolation. Since domain decompositions, which do not maintain data balance, perform less efficient for the quality control, they have not been reimplemented and are not further considered.
To explain why the performance of the cyclic decomposition was not as good as originally expected, Figure 6.14 displays the times each processor needs to complete its task. For

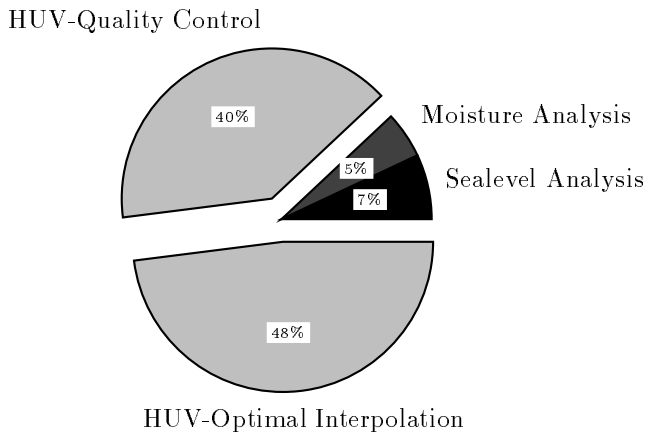

Figure 6.12: *Fractions of the computational intense parts of the sequential optimal interpolation algorithm for the Versions 1.2, and Dataset B.*

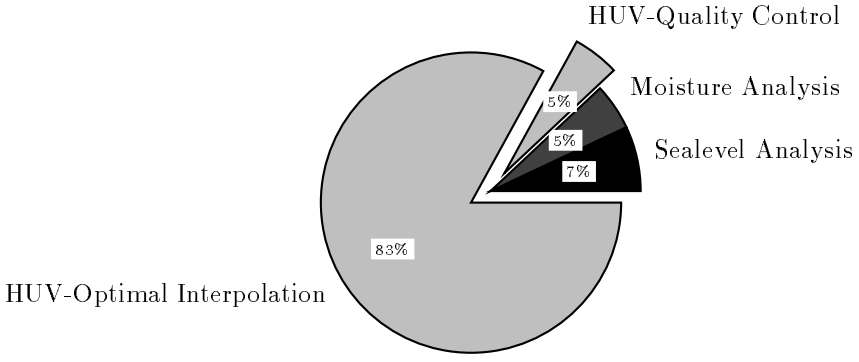

Figure 6.13: *Fractions of the computational intense parts of the sequential optimal interpolation algorithm for the Versions 2.0, and Dataset B.*

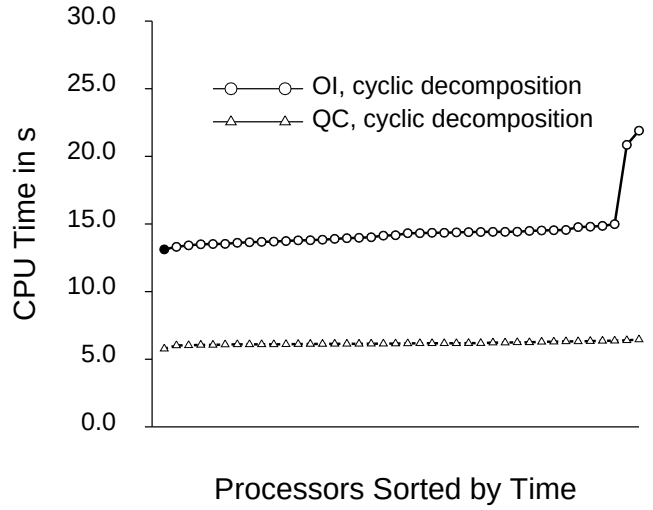

Figure 6.14: *The CPU times for the HUV quality control (QC) and the optimal interpolation (OI) while using 40 processors. Both use a cyclic domain decomposition.*

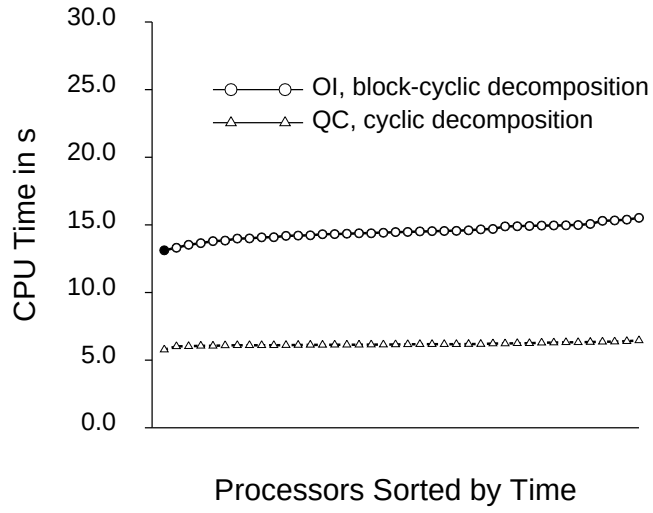

Figure 6.15: *The CPU times for the HUV quality control (QC) and the optimal interpolation (OI) while using 40 processors. The OI uses a block-cyclic domain decomposition.*

Table 6.4: Comparison of the runtime of the versions 1.2 and 2.0. Different decompositions on 40 processors are used.

Version	1.2	1.2	2.0	2.0	2.0
domain	Cart.Grid	observation	observation	minivolume	minivolume
Decomposition	striped	bisection	cyclic	cyclic	bock-cyclic
databalance	no	yes	yes	yes	yes
Quality control					
Time in s	87.3 s	56.3	6.5	-	-
Speedup	1.0	1.6	13.5	-	-
Optimal Interpolation					
Time in s	55.5 s	55.4	-	21.5	15.5
Speedup	1.0	~ 1.0	-	2.6	3.6

better visualization, the times for the tasks assigned to the processors are sorted by their value. Clearly, the timings for the two processors are significantly different. These processors contain the polar regions which contain a larger linear system of equations to be solved.
Assigning fewer minivolumes to the processors containing the polar regions does not solve the problem of decreased efficiency, because the calculation time is dominated by the time to conduct the calculations at the poles. The algorithm is not scalable. Maximal 74 processors can be used. The time over the poles are 7.39 seconds and 8.05 seconds, respectively. The average time over the rest of the minivolume columns is approximately 0.5 seconds.
Decomposing the domain over the poles in vertical layers, generates smaller tasks. Hence, better load balance is achieved (see Figure 6.14). The range of the computational load is displayed in Figure 6.16, with the help of a Box-Whisker diagram.

6.5 Comparison of the Different Domain Decompositions

So far, the parallel algorithms for the assimilation system have been evaluated only while using 40 processors, in order to explain major properties of the algorithm. Since the cyclic decomposition (applied on the observation domain) achieves very good results for the quality control, it remains to find good decompositions for the optimal interpolation.
The overall performance for different domain decompositions for the optimal interpolation

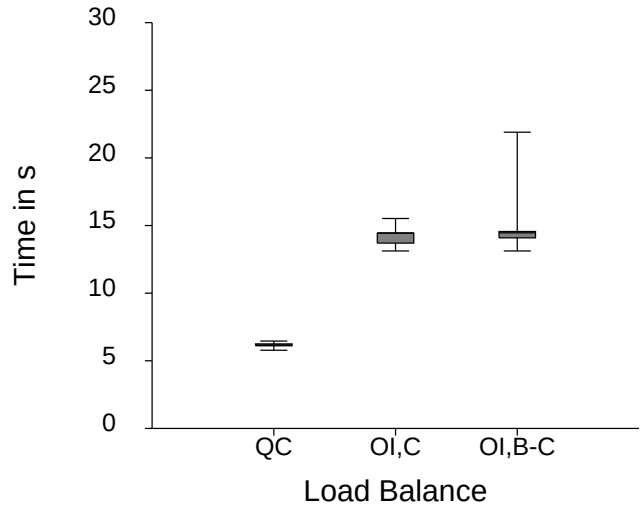

Figure 6.16: *The range of the CPU times for the HUV quality control (QC) and the optimal interpolation (OI) while utilizing 40 processors and using the cyclic (C) and the block-cyclic (B-C) decomposition.*

is shown in Figure 6.17. A closeup is depicted in Figure 6.17, giving the results for smaller numbers of processors (up to 25).
In the figures, the abbreviations S,B,C, and D, are used to distinguish the different domain decompositions. We specify

S to be a placeholder for the *striped* decomposition,

B to be a placeholder for the *block* decomposition,

C to be a placeholder for the *cyclic* decomposition, and

D to be a placeholder for the *dynamical* decomposition,

The block cyclic decomposition, as introduced in Chapter 4 can be extended to any one of the regular decompositions. The first letter indicates that the domain is decomposed in a number of blocks. The second letter indicates which strategy is used to decompose the block not containing the polar region. The third letter is used to indicate which strategy is used to decompose the vertical levels over the polar blocks. Distinguished are:

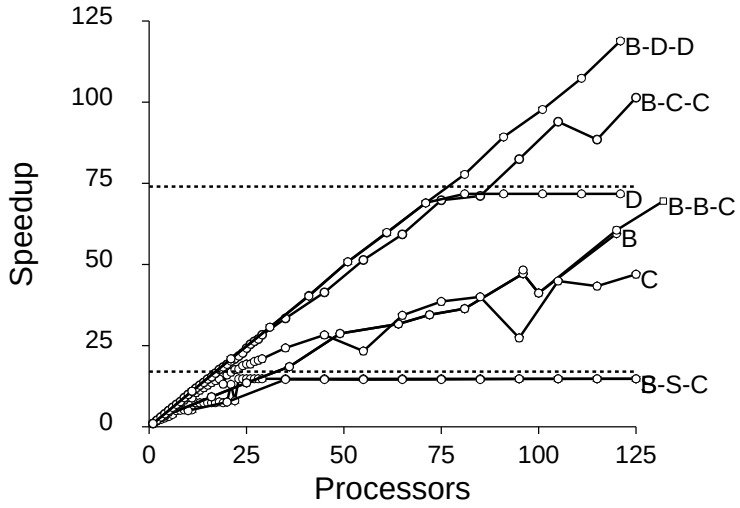

Figure 6.17: *The comparison of the different load balancing strategies for the HUV optimal interpolation for larger numbers of processors. The legend is given in the text.*

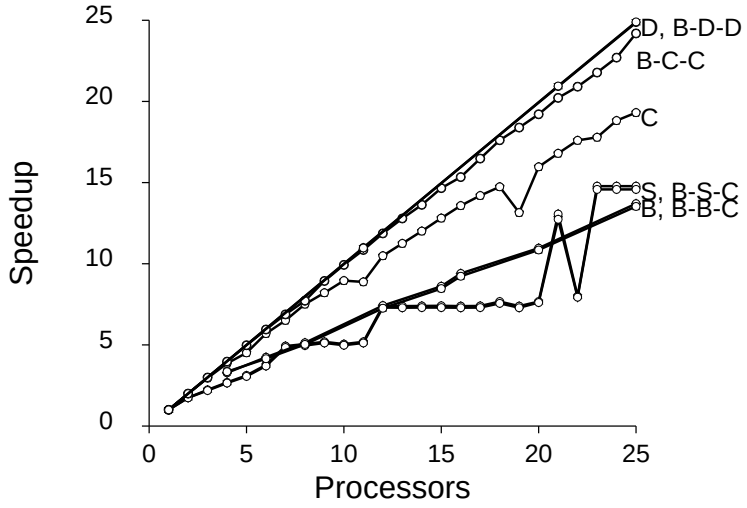

Figure 6.18: *The comparison of the different load balancing strategies for the HUV optimal interpolation for smaller number of processors. The legend is given in the text.*

B-S-C – the block-striped-cyclic decomposition, the vertical levels of the poles are cyclicly distributed, while the rest of the domain is decomposed in latitude stripes,

B-B-C – the block-block-cyclic decomposition, the poles are cyclicly distributed, while the rest is decomposed in blocks,

B-C-C – the block-striped-cyclic decomposition, the vertical levels of the poles are cyclicly distributed, as well as, the rest of the domain, and

B-D-D – the block-dynamic-dynamic decomposition, everything is distributed dynamically.

For small numbers of processors, we obtain the following results:

1. The dynamic B-D-D decomposition performs the best.
2. The static B-C-C decomposition performs almost as well.
3. The simple cyclic C decomposition outperforms all striped and block decomposition (S,B) algorithms.
4. Geographical load imbalance on the spatial domain is the reason for the difference between the cyclic and the striped and block decomposition.
5. For the striped decomposition an upper boundary of 15 processors exists, after which no performance increase is noticeable. This is due to the fact that for our test case the distance between the minivolumes in latitude direction is larger than the assigned latitude stripes.
6. Since the computational tasks for a small number of processors are large, the vertical decomposition over the poles does not lead to big improvements. Thus the curve for the block decomposition is almost the same as the one for the block-block-cyclic decomposition. The same is valid for the two striped decompositions.

Due to the simplicity[1] of the cyclic decomposition and the ability to transfer it into other programming paradigms, we recommend to use this decomposition instead of the dynamic load balance strategy.
For larger numbers of processors, we obtain the following results:

[1] One of the requirements of NASA DAO

1. In case, the polar regions are not decomposed in horizontal layers, the runtime is dominated by the calculations performed at the poles. An upper boundary of 74 processors can be used before the calculation time is dominated by the poles.
2. The poles should be decomposed vertically.
3. The static B-C-C decomposition performs well, even with high numbers of processors.
4. As for small numbers of processors, the dynamic B-D-D decomposition performs the best.

In case even more processors are used, the performance of the dynamic load balance algorithm will decrease. This can be changed while using a hierarchical load balance algorithm or an algorithm with decentralized control.
At this point of the parallelization, other parts (like the input and output) become dominant and consume a considerable amount of calculation time in contrast to the parallelized parts of the algorithm. It is more cost effective to spend time on the restructuring and recoding these sequential parts of the assimilation system, then spending time to improve the well parallelized quality control and optimal interpolation. Again, Amdahl's law motivates this.

Chapter 7

Metaproblem, Metacomputing, and Dataflow Concept

7.1 Problems - Theory - Solution - Resources

Humans have the abilities to detect and solve problems. Solving a problem is a challenging task. Generally, it involves the formulation of a theory and strategy to find a solution based on the availability of limited resources.
One can abstract the dependencies between theory, problem, resources and solution, as shown in Figure 7.1. Multiple dependencies exist between problem, theory, solution, and resources. Three obvious dependencies are:

1. To develop a solution, a theory and resources are necessary.

2. Resources are necessary to abstract the problem and formulate a theory for solving the problem. Developing a theory poses a problem in itself, and requires a solution.

3. To assign resources, in order to develop a solution or a theory, can pose a problem.

Often, it is necessary to subdivide a problem into smaller problems because the subproblems might be easier to solve. Solving each of the subproblems and combining the result will lead to a solution for the original problem. Theories, and solutions for each of the subproblems, have to be established in order not to distort the original problem. Usually, the sum of all

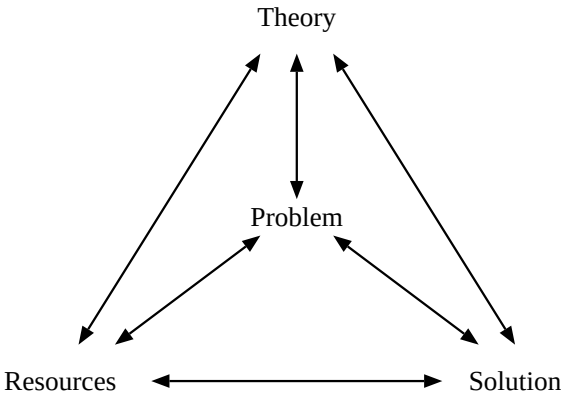

Figure 7.1: *The dependencies between problem, theory, resources and solution.*

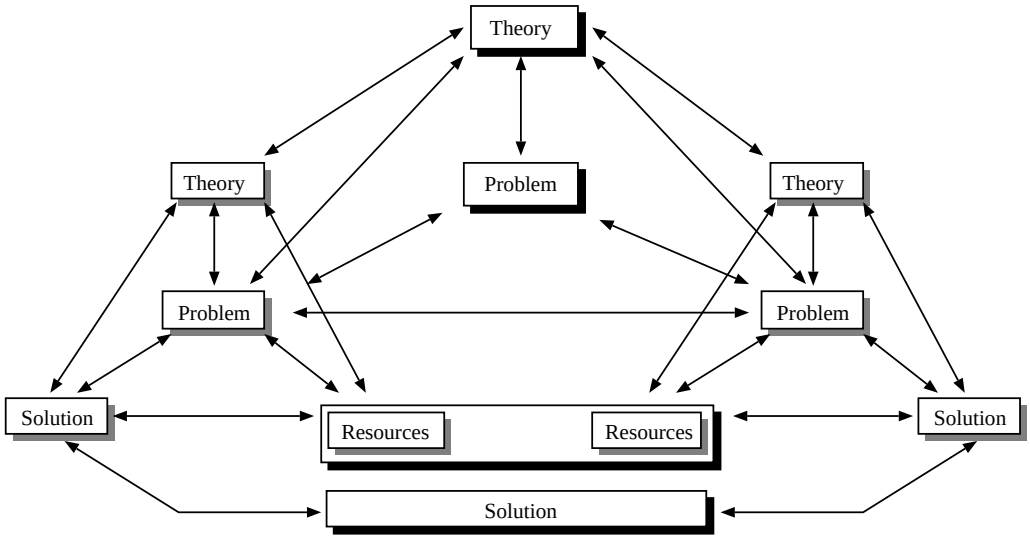

Figure 7.2: *The problem pyramid.*

resources used for the subproblems can not exceed the overall available resources. Figure 7.2 shows an example of some of the dependencies, in case the original problem is split into two subproblems.

Dealing with limited resources introduces complex dependencies between the solution of one subproblem, with another subproblem. One way to relax this situation, is to provide a large number of resources. Unfortunately, many restrictions on the availability of resources are present in real life problems and their solutions. This is especially true for high performance computing problems.

For a computational task, a theory and a model to solve the problem have to be established. When the resources are too restrictive, a good solution might not be found. Dealing with the problem of limited resources is an integral part of high performance computing

7.2 Grand Challenge Problems

A special class of problems are the *grand challenge* problems. For grand challenge problems, the current computational resources of one single computer are not sufficient to achieve a solution of the problem in acceptable time.[1] State-of-the-art supercomputers are in use and under development to find solutions to subproblems of the original grand challenge problem. The combination of the solutions obtained on one or more machines are combined to obtain an overall answer.

The considered solutions for grand challenge problems are bound by the limited resources. Most importantly, large memory and many processing units of high speed are required. Since both resources are of limited availability, compromises in the program design have to be considered.

7.3 Metaproblems

From the computational and computer science point of view, a classification of *complex problems* is introduced in [54, 48]. Applications for solving complex problems are distinguished. The applications are classified into 5 classes:

[1] Time can also be considered as a resource.

Synchronous applications tend to be regular and are characterized by algorithms employing simultaneous identical updates to a set of points.

Loosely synchronous applications are iterative or time-stepped, but unlike synchronous applications, employ different evolution(update) procedures which synchronize macroscopically.

Embarrassingly parallel applications employ complex algorithms, but can be parallelized because the evolution of different points is largely independent.

Asynchronous applications are hard to parallelize problems with no natural algorithmic synchronization between the evolution of linked irregular data points.

Metaproblems are a hybrid integration of several subproblems of the other four basic application classes.

7.4 Metacomputer

The metacomputer is a natural evolution of existing computing technology. Often, Flynn's well known classification is used to categorize parallel computers according to the instruction and data stream[39]. Here, the MIMD - Distributed Memory (Multiple Instruction Multiple Data) architectures are of special interest (Figure 7.3). It is important to note that the memory is an integral part of each processing unit. The memory is local to each processing unit. Data is exchanged via message passing. Multiple Instructions can be executed on Multiple Data (MIMD). Examples of real MIMD computers are numerous (SP2, Hypercube, and many more)[21, 44, 108]. The intercommunication between the processing units is achieved via a high performance interconnection network.

In contrast to stand-alone MIMD supercomputers, heterogeneous computing environments consist of a number of different processing units. Often, the processing units are off-the-shelf workstations connected via a fast network (Ethernet, ATM). The user of the heterogeneous computer is aware of the different machines and has to incorporate the knowledge about the distributed resources in the design of the program. Operating systems are available and under development to provide a better and automated resource management[105, 72].

The new development in computational and computer science is to

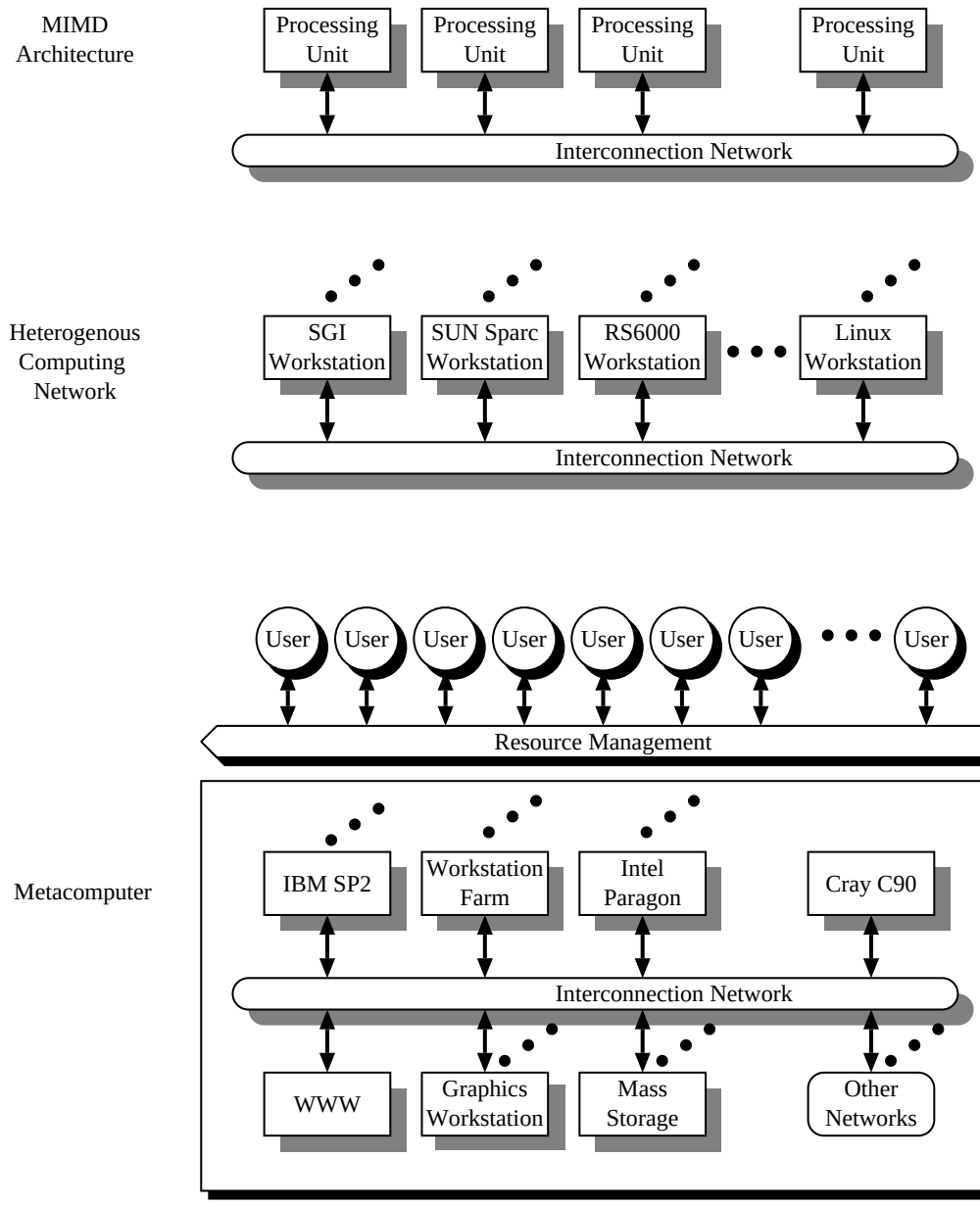

Figure 7.3: *The MIMD Architecture, a heterogeneous computing network, and a metacomputer.*

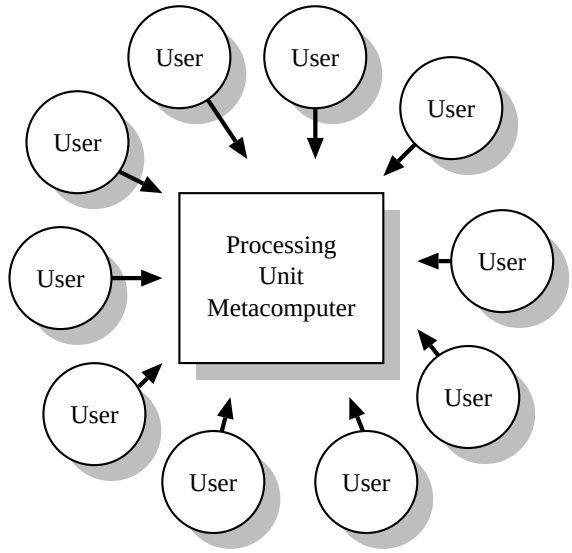

Figure 7.4: *Metacomputer from the user point of view.*

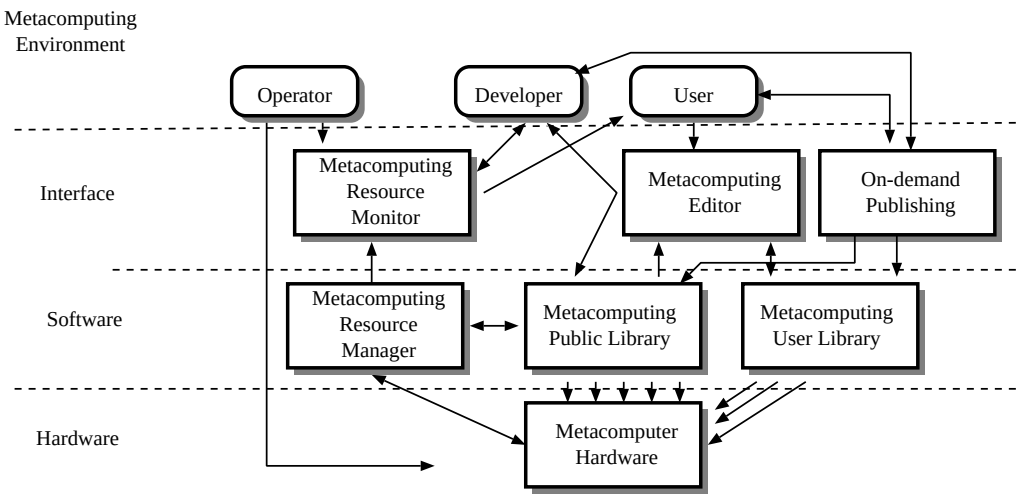

Figure 7.5: *Essential parts of a metacomputer for operators, developers and users.*

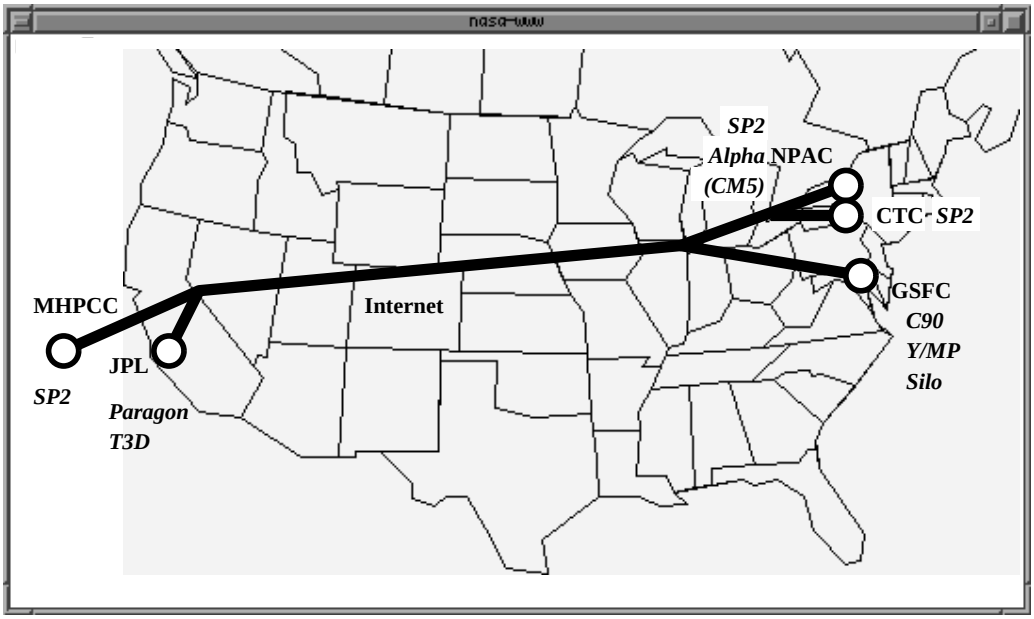

Figure 7.6: *The geographical distribution of the metacomputing components as utilized in the NASA Four Dimensional Data Assimilation Project.*

1. add more, and inherently different resources, to a heterogeneous computing environment,

2. add supercomputers as processing units to the network, and

3. simplify the programming and usage of the increasingly more complex computing platform.

This collection of state-of-the-art computers and technology is referred to as a *metacomputer*[99, 92]. For the user, the metacomputer acts like a single computer. Ideally, a user should not know from which hardware the metacomputer is built upon. The user should be able to formulate the solution to a problem in his or her favorite programming paradigm. The program, is then executed in the *black box* metacomputer, with minimal supervision from the user (Figure 7.4). A clever resource management strategy utilizes the available resources, best suited to execute the program. The resource manager is an essential part of the metacomputer. Due to its generality, a *metacomputer* provides the computational resources to

solve metaproblems.

Figure 7.3 shows an example of the hardware components of a metacomputer. These hardware components have all been utilized in the NASA Four Dimensional Data Assimilation Project. The physical location of the hardware components of a metacomputer can be in the same building, in different cities, and even on different continents. The resources available for the NASA project are distributed in the United States (Figure 7.6) and include the island of Hawaii.

To employ a functioning metacomputer, it has to be installed, maintained and updated. Thus, besides users, operators and application developers have to be distinguished.[2] This leads to the more complex picture of a metacomputer, as depicted in Figure 7.5. The distinction between a user and a developer is based on the fact, that a user can not easily incorporate new software into the publicly available software library.

A metacomputer will not function correctly if one does not provide appropriate tools for using, extending and monitoring the metacomputer. The inclusion of these tools to the metacomputer provides the *metacomputing environment*. As depicted in Figure 7.5, a metacomputer environment consists of a hardware, a software and an interface layer. The success of using metacomputers will depend strongly on the metacomputing environment, e.g. the interface layer. The following parts should be incorporated in the design of a metacomputer environment:

Hardware Layer

- **Processing units** are usually workstations, supercomputers, and PC's.
- **Communications Hardware** or **Network**, consists of high and low bandwidth networks, like Ethernet, ATM, and token ring.

Software Layer

- **Communications Software** is the high level software which allows the communication over a network, e.g. TCP/IP, sockets, MPI, and http.
- **Resource Managers** are responsible for distributing the resources of the metacomputer appropriately. This is done with the help of load balance strategies, partitioners, task schedulers, domain decomposers, and batch operating queues.

[2] In literature a user is often referred to as a developer.

Interface Layer

- **User Support Facilities** help the user to facilitate the metacomputer. It provides scientific visualization, data flow editors, and programming languages.
- **Metacomputing Public Library** contains the source code or the precompiled code for different machines in different programming paradigms. Performance data is stored with the library.
- **Metacomputing User Library** has a similar function as the public library, but is organized and maintained by the user instead of being publicly distributed with the system.
- **Operator Support Facilities** help the operator to control and supervise parts of the metacomputer. It includes network monitors, utilization and load balance analysis monitors, tools for authorization and security, tools for the incorporation of new resources, and tools to perform trouble shooting.

The main requirements users, operators, and developers pose to a metacomputing environment are: ease of use, portability, maintainability, expandability, reusability, and fault tolerance.

In the following sections, we will concentrate on the specification of tools which enable the use of a metacomputing environment. The motivation for these tools arose from the practical experience gained at NASA GSFC and at NPAC with diverse grand challenge applications and software environments[115, 118, 122].

First, we will concentrate on the developer and user. The developer provides the software resources necessary for an application to be executed on the metacomputer. The user applies the tools developed, and constructs its own applications, which are not necessarily publicly accessible to others. The distinction between users and developers is useful because of their different security aspects.

An appropriate program editor should provide an easy to use interface to a program library for the users and developers. The automatic maintenance of resources should be supported, to ease the task of integrating existing computational facilities into the metacomputing environment for an operator.

Since the success of the environment will be dependent on the interfaces provided, a top down approach is followed starting from the graphical user interface.
Other projects [52] pursue a bottom up approach. The SNAP project pursued at NPAC will provide an infrastructure of low-end, but widely usable, applications and technology. One of the components planned, is the development of an interface to automatically control computations to be distributed on the resources of the WWW. Messages can be exchanged with the different computers on the WWW, thus forming a truly heterogeneous computing platform. The GUI environment introduced here, will be able to utilize the low-end interface and incorporate it without the users knowledge. The interface does not change. Thus, the bottom-up approach of SNAP, and the top-down approach for designing the user interface, are viable additions to each other. The lessons learned form the GUI environment can be used in the forthcoming WebFlow project at NPAC[46, 45].

7.5 Motivation and Requirements for the Metacomputing Environment

The motivation for the development of a metacomputing environment arose from the rather complex problem of parallelizing the NASA Four Dimensional Data Assimilation System. Insight into the operation and resources of the DAO have been gathered during an extended research period of 6 months, at GSFC. The experience gained from parallelizing the first version of the code, and the interaction with the atmospheric scientist, revealed that there is a need to simplify the access of parallel computing. This is also true for other fields in scientific computing.
After analyzing the departmental resources, it became quickly clear that only a limited period of time is available to teach parallel programming paradigms to the scientists. This also includes the practical use of a parallel system. Using graph flowcharts helped to explain certain aspects of the different programming paradigms. Atmospheric scientists are used to the data flow concept, because many concepts in atmospheric dynamics are illustrated with the help of directed graphs. In atmospheric dynamics, parallelism is implicitly represented in these graphs. Thus, it is most natural to use the graph concept to express parallelism and formulate parallel programs.
This is not a new idea and has been incorporated in many programming languages, as well

as, visual programming interfaces. An example for a task parallel programming language can be found in [63, 41]. Here, the design of a parallel program is often initiated by deriving the tasks and their dependencies to obtain a task graph. The task graph is then transformed to a textual form by hand.

The new idea is to use the concept of dataflow[68] in a more general way. With the help of dataflow concepts, many different programming paradigms can be combined into one program. The information about which programming paradigm is used, is hidden from the users, thus enabling a uniform interface for multiple programming paradigms. Besides the support for explicit message passing, task parallelism and data parallelism should be supported. Providing the dataflow programming paradigm, enables the programmer to transfer the flow representation into the different parallel programming paradigms.

An important part of the environment is to guarantee that previously written subroutines and programs can be integrated into the metacomputing environment. They can be sequential programs, as well as, parallel programs and can follow different programming paradigms. It is essential to support the integration of programming languages like Fortran, Fortran90, C, C++, and others, to achieve acceptance in the scientific community. Integrating HPF[53, 40] will be of advantage in the future.

Furthermore, the metacomputing environment should enforce more strict software engineering techniques, in order to force some programmers to provide a documented code.

For now, we summarize the requirements of the graphical user interface, evolved from grand challenge applications:

Paradigm Flexibility is needed to support multiple parallel and sequential programming languages, as well as, multiple programming paradigms. This includes, e.g., sequential programs, task parallel programs, and data parallel programs.

Expressiveness is needed. Supporting multiple programming paradigms increases not only the flexibility, but also the expressiveness. In addition, the graphical display of functions increases the readability and expressive power of a program, in contrast to its textual form.

Ease of use is needed to motivate the user to learn a new programming environment which has more expressive power than the tools she or he used before.

Portability is needed to incorporate the large amount of different hardware, from which the metacomputing environment is built.

Extendibility is needed to include new procedures in the library, accessible by the user. A user should be able to maintain its own library.

Restrictiveness is needed to incorporate software engineering standards for the documentation of the code. A guideline should be provided on how to document a given module. Input and output parameters should be described. If possible, resource requirements about space, time, and complexity should be specified.

Execution Flexibility is needed to update of the environment at runtime. This is done on two levels. A) The update of graphical icons to represent the state of the calculation. B) The modification during the program run to incorporate new solutions and programs to the actual running application.

A more controversial requirement will help in using the system in a real life setting:

Resource Restrictiveness is useful to specify a particular mapping of the program to a given hardware configuration.

The last requirement usually contradicts the concept of a "black box" metacomputer, where the resources are distributed, without the knowledge of the user, via the help of the resource manager. Nevertheless, it is important to note, that certain resource and paradigm restrictions are known to the the user and developer, in advance. The knowledge of mapping a particular module to an appropriate machine can improve the overall performance of the program drastically, because it provides viable information to the resource manager. If the information is not available, the resource manager will make the decisions independently.
Next, the important concept of dataflow and its use in literature is described in more detail, because it is integral part of the metacomputing environment.

7.6 Dataflow Concept

The term *dataflow* has different meanings depending on the context in which it is used [68, 11]. In software engineering, it refers to the flow of information between data processing units.

Already in 1966, the term dataflow has been used in the context of parallel computing[71]. Besides the theoretical acceptance of dataflow models in the late 60's and early 70's[70], dataflow had also an impact on computer architecture designs[30, 31].
In general, a program is evaluated with the help of a computing model. In the case of a *von Neuman* machine, the computing model is control-flow oriented. A program is evaluated step by step in a processor. The control is transfered from one command to the next command. The operands of the commands are fetched into a fixed memory location (the registers). Then, the command is executed on the contents of the registers.
In contrast to the control-flow model, a data flow model is based on the flow of data. The program is specified in a data-dependency graph or data flow graph. Usually the nodes represent operations, while the edges represent dependencies between the operations.
The main difference between control-flow and dataflow computing models, is the fact that the program execution in

1. control flow programs, "corresponds to the instructions in motion operating on data at rest",

2. dataflow programs, "corresponds to data in motion being processed by operations at rest[68]."

Several, dataflow models are distinguished in literature[68]. In the data-driven dataflow model, values(tokens) are produced and consumed by the nodes of the dataflow graph. A token is generated and sent to an output edge. An operation associated with a node can only *fire* when all its input edges have a token.[3] This simple *statical dataflow model* can be expanded with the help of a queue for each edge. In addition, each token gets an identification tag. A node fires as soon as it recognizes tokens with the same tag on its input edges. This dataflow model is known as the *classical dynamical model*. The dynamical model is necessary to express asynchronous parallel computational tasks. It allows one to use loop parallelism and recursive parallelism, dynamically at runtime.
A slightly different view of dataflow is sometimes used in parallel computing. Here, a process graph in which data flows from one processor to the next, is used to express parallelism. Under the abstraction that its inputs input data are consumed and its output data are produced,

[3] In contrast to the data driven model, is the demand driven model, which fires only when there is a demand for the result, and all tokens at its input edges are known.

this graph is referred to as a dataflow graph. Such a dataflow model can be implemented in several different ways. The easiest abstraction model uses synchronous channels between the processing nodes, as introduced in [19]. Nevertheless, we find this abstraction model too restrictive because it guarantees only synchronized flow of data between the processing units. Thus, it is similar to the classical static dataflow model. To increase the amount of parallelism, a *dynamical data flow model* can be achieved via the introduction of *asynchronous channels*.

An asynchronous channel has the ability to transmit a number of messages (tokens) without maintaining the order on the arrival of the tokens. A (time) tag is attached with each token to identify tokens with matching tags for the operation. In case the number of tokens available is too large, the producer of tokens has to be stopped. When no matching tokens are in the input buffers dead-lock occurs. The static data-flow model avoids this problem by allowing only one token at a time at any input edge. Many solutions to a possible deadlock prevention are possible. Fortunately, the application motivating the development of the metacomputing environment does not cause any deadlocks, even if asynchronous channels are used.

Representation

Usually, the nodes in a software engineering dataflow model are functions. In many graph representation only the functions are depicted as nodes. Data is implicitly forwarded via the edges. Thus, in these data flow models a memory location or an assignment to a particular unit is not specified directly but implicitly.

Since the data mapping and domain decomposition are essential parts of parallel programming, it is desirable to represent the processes, as well as, the data (memory) with nodes. The values of the data stored at the memory nodes can be forwarded, via the messages, to the process node. Thus, a data node can be viewed as a special process node with the identity function.

This graph is referred to as a *data-process-flow graph*. The processes have to be executed on real machines, and the data has to be mapped into the memory of the machines. A mapping between the data nodes and the process nodes to the real machines is necessary. We call the graph, which includes the mapping onto a real machine, the *enhanced data-process-flow graph*.

One problem of dataflow programming models, is the complexity arising in order to express

the parallel program in written form. Firing rules and matchings have to be generated.
To introduce the usually rather unfamiliar concept of dataflow programming to non computer scientists, will cause problems. A graphical program editor, hiding most of the details for specifying matching and firing rules, is most useful. Thus, the program editor should not only allow the specification of data and process nodes, as well as, their flow dependencies, but also the specification of the mapping of data and process nodes to a real computer architecture. This visual dataflow editor is introduced in the next chapter.

Chapter 8

The Interface and Software Layer of the Metacomputer

In this chapter, the components of the interface layer and software layer of the metacomputing environment are explained (Figure 7.5). A prototype implementation of a graphical user interface (GUI) for the metacomputer developer and user, is introduced. It is referred to as the *metacomputing editor*. The input specified with the help of the metacomputing editor is based on the dataflow concept. Besides the metacomputing editor, the resource manager and the computing libraries are described. On demand publishing and a resource monitor provide other interfaces to the metacomputer. Advantages and disadvantages of the existing prototype are analyzed, and future improvements are listed.
The metacomputing editor, introduced in this chapter, can be used for multiple tasks. Even though the tasks are quite different in nature, the same principles are used to visualize them with the help of graphical concepts. Distinguished are

1. tightly coupled metacomputers,
2. loosely coupled metacomputers,
3. and metacomputers which are a combination of both.

The ultimate goal is to incorporate tightly and loosely coupled metacomputers in one environment. Currently, their usage is separated.

A *tightly coupled metacomputer*, is a metacomputer which makes use of software technology, enabling the fast communication between processing elements. Examples are message passing programs generated for MPI, PVM, or others. The parallel programs generated with the tightly coupled metacomputing environment editor are usually of medium grain. The resource allocation for programs generated in such a way, involves the determination regarding which processors or workstations are best suited for the program execution. Grand challenge applications have a definite need for such an application development environment, to reduce the software engineering costs involved.
A *loosely coupled metacomputer*, is a metacomputer which combines the usage of several supercomputers, or workstation clusters. A process executed on one of the machines is called job. Message exchange between the components is done with the help of files. Instead of using blocking send and receive commands, asynchronous communication is established with the help of a probe command. The probe command checks for the existence of the file or place holder associated with the message. This is very much like the concept known from MPI[83, 100].
This kind of system is useful, when

1. the messages to be sent are infrequent,
2. the time between two messages is assumed to be long,
3. the messages to be sent are long,
4. the processes use a lot of CPU time, and
5. the message passing overhead is small.

Both tightly and loosely coupled problems occur in the NASA project, and are addressed separately.
The special requirement for a tightly coupled metacomputer is the speed of communication and computation. The special requirements for a loosely coupled metacomputer is the efficient usage of the computing resources. The communication time is important, but is usually much smaller than the computation time. Communication is usually a secondary issue. Fault tolerance, has an increased importance in such a distributed (meta)computing environment.

8.1 Metacomputing Editor

In the following sections, the GUI for the tightly and loosely coupled metacomputer are introduced. The interface shares common features and uses the same front-end of the GUI.

8.1.1 A Tightly Coupled Metacomputing Environment

To fulfill the requirement of ease of use and flexibility, the metacomputing editor allows the specification of a parallel program with the help of a visual graph representation[120]. Nodes and edges of the graph are drawn in an interactive window. The functions to be assigned to the nodes and edges can be specified in an arbitrary programming language. The environment is able to generate stand alone programs, but also program modules which can be reused later. They are included in a module library. If the user, is an authorized developer, the module can be integrated in the standard set of modules accessible by other users. Otherwise, they are accessible with the help of a WEB server which is write accessible by the user and read accessible by others. The programs and modules which are specified with the metacomputing editor are compiled on a set of specified target machines. The visual system forces the programmer to use a more strict and regulated method of programming, in regards to providing documentation of the modules and the programs. Hence, the code produced by the environment is ultimately more easy to maintain, and problems existing with poorly documented legacy codes are avoided.
Even though the original environment has been first designed in 1993 for the NASA Four Dimensional Data Assimilation System, it will be useful for many other application programs. During the parallelization of grand challenge problems, several steps could be distinguished. The steps, which are not only typical for grand challenge problems, are:

1. Design the global program structure, keeping in mind the parallel nature of the problem.
2. Specify the function of the blocks which build the global program structure.
3. Convert program blocks which are of concurrent nature, into parallel blocks.
4. Map the parallel and sequential blocks on a real architecture.
5. Run the parallel program and observe performance statistics.

Following this program design, the visual editor should support all phases of the parallel program development.

The Figures 8.1-8.6 show the practical use of the metacomputing editor for the NASA Data Assimilation System. Figure 8.1 displays the logical division of the objective analysis algorithm.

Data is generally shown in rectangles, while processes or tasks working with data are displayed in circles. Dependencies between data and tasks are displayed with the help of directed 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 8.2. A parallel process is visualized with multiple circles, while data distributed in an exclusive way is visualized with multiple rectangles. An example for such distributed data is the block distribution, as known from HPF.

The definition of **data objects** *flowing* between **process objects** can be done graphically in a selection window containing all available data types. In addition, a straight forward textual representation of the messages is possible, and can be specified with a standard text editor.

An example for the textual representation of a FORTRAN data object, is shown in Figure 8.8. Here, a simplified data structure from the NASA code is displayed. Due to its simplicity, the representation of data objects for different languages is very close to the language standard. The native names for datatypes, as well as, block statements are used to support the definition.[1] To allow custom designed datatypes to be elements of a data object, it is advisable to define them previously as data objects with the help of the metacomputing editor.

In general, the definition of data objects is similar to a RECORD, as known from several Fortran77 extensions, Fortran90, and the struct command in C. The definition of a data object will generate the necessary routines allowing communication between the data objects and the process objects. The user will implicitly use these commands while drawing arcs from one process to the next process.

Figure 8.9 shows the usage of a data flow object in a process definition. The indirection of the data flow is marked with the special keywords IN_DATA and OUT_DATA. The keyword PROCESS is introduced to distinguish between subroutines and processes. A dataflow pro-

[1] For C these are the braces {, }, and int, float, double, and many more.

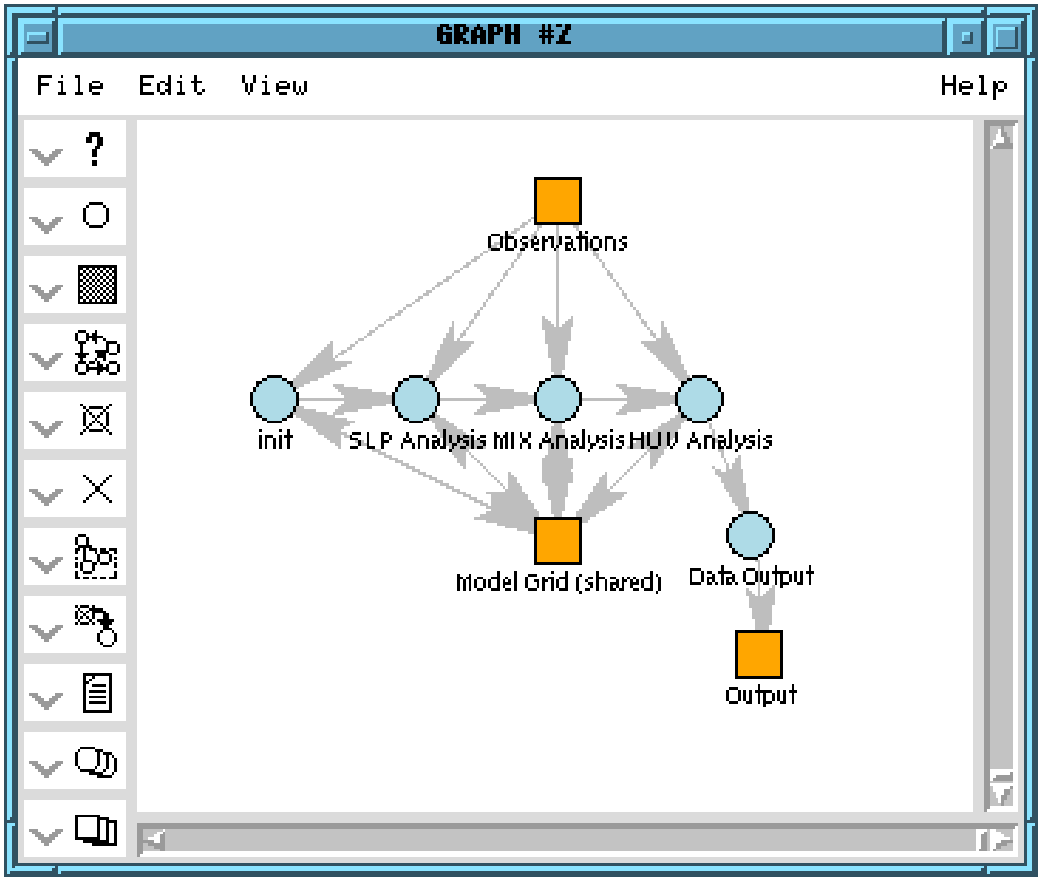

Figure 8.1: *The window shows the building blocks used in the global program structure (tightly coupled metacomputing program).*

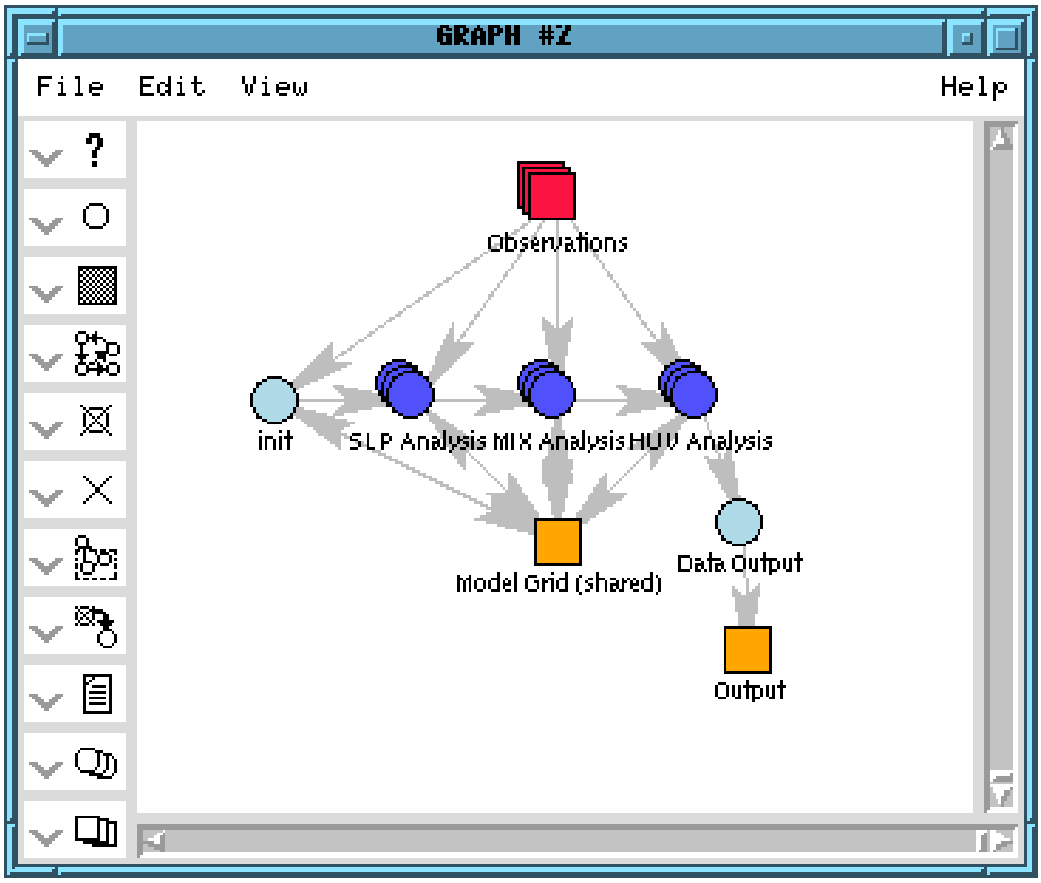

Figure 8.2: *The window shows how the program is represented after the parallel program blocks have been introduced.*

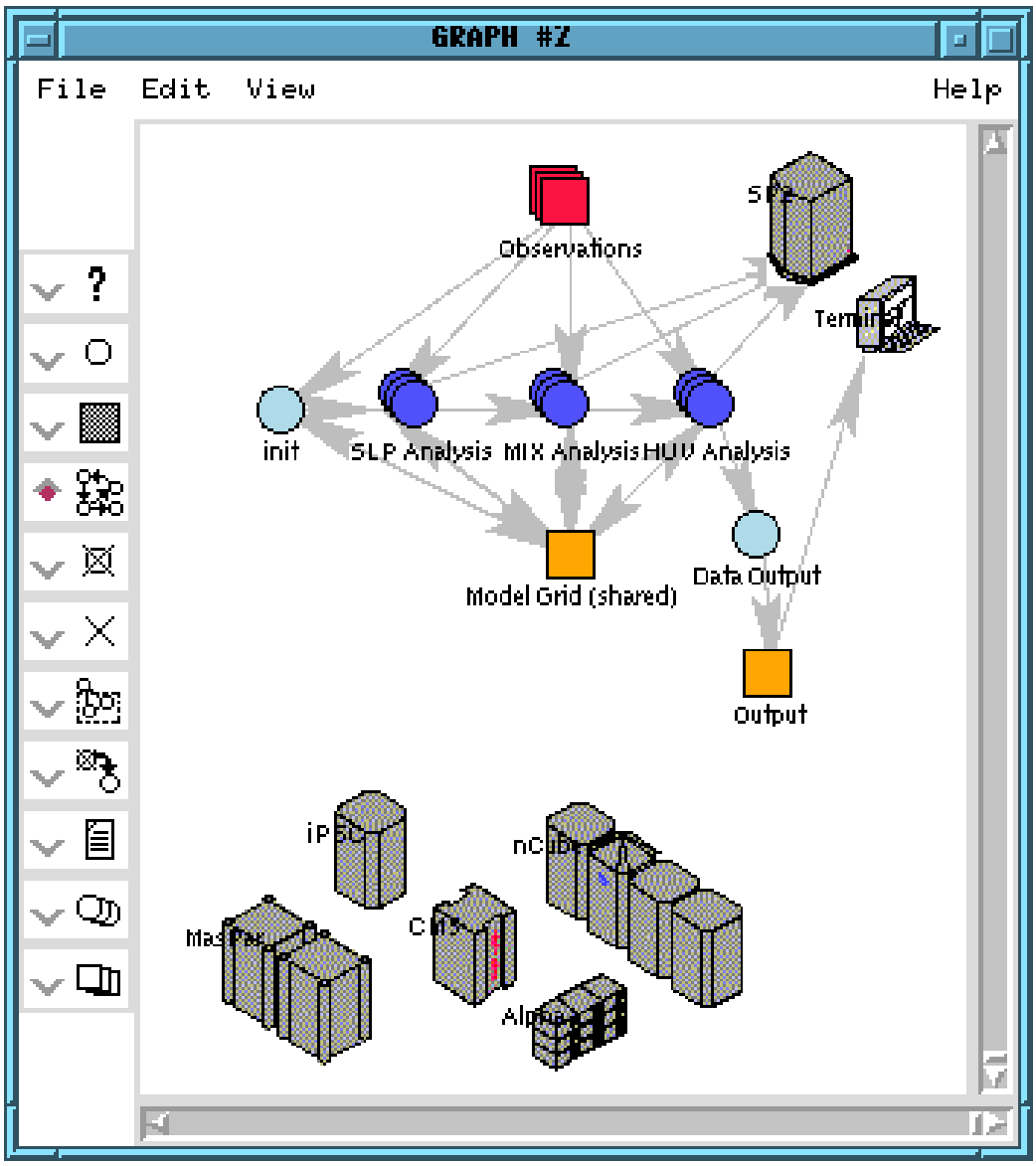

Figure 8.3: *The window shows the selection of the machines participating in the execution of the program.*

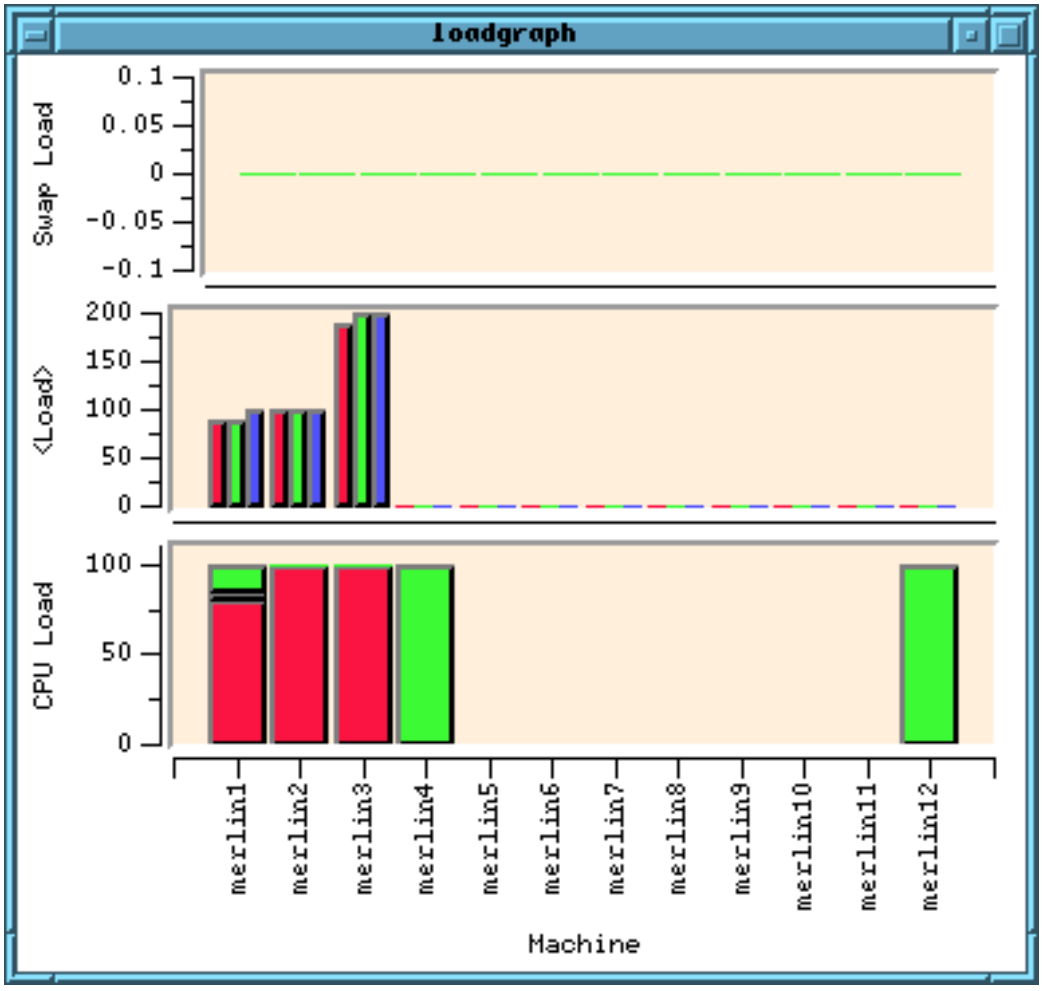

Figure 8.4: *The window shows the load meter to control dynamic load balancing while executing the code.*

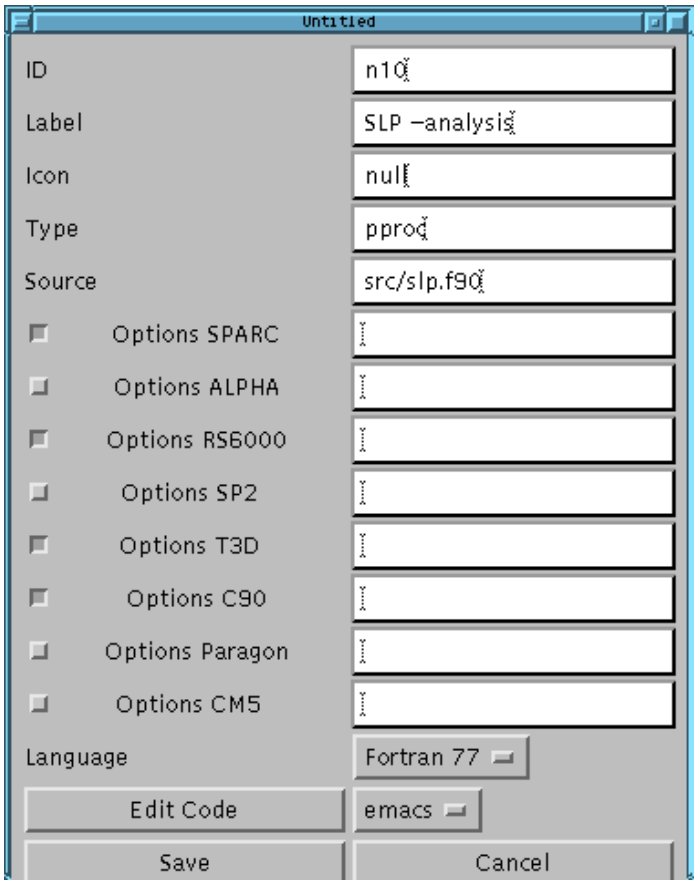

Figure 8.5: *The window which specifies the machines on which the module should be available, where the source code is located, and what graphical representation the node should have.*

Figure 8.6: *The module selection with a listbox.*

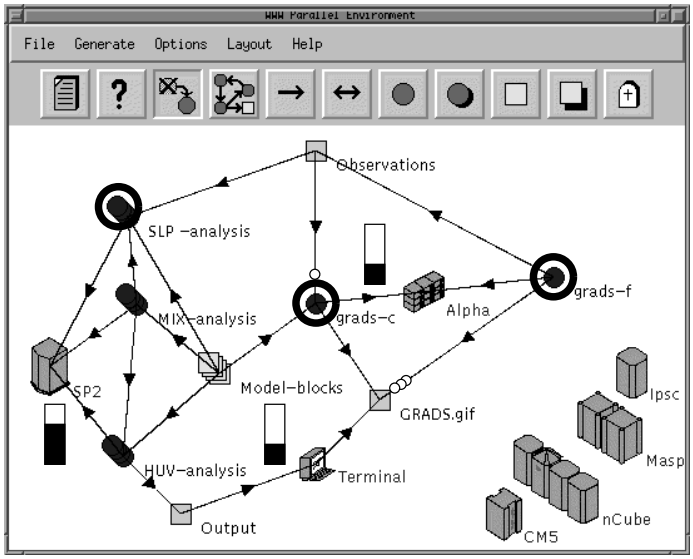

Figure 8.7: *A snapshot from the running application, augmented by the current load on each machine and the processes currently active (the ones marked with big circles).*

```
DATA OBJECT OBSERVATIONS

    INTEGER NoOfObservations;
    REAL x(NoOfObservations);
    REAL y(NoOfObservations);

    REAL temp(NoOfObservations);
    REAL pressure(NoOfObservations);

END DATA OBJECT
```

Figure 8.8: *Definition of data able to flow between process objects. The data object is a simplified data object as used in the NASA project.*

cess is an independently executed program which waits for the instantiation as soon as its parameters are available. Thus, with only very limited extension to the original sequential programming language, in this case Fortran, task parallel programs can be defined easily and naturally. More interesting problems can be thought of, while sending dynamical data structures as found in irregular problems. For future research, we point out that the extension of the data flow concept with actual programs as data, will enable the distribution of programs similar to the distribution of data. Special care has to be taken in order to solve security and byte order issues. By choosing Java as the language, in which the processes are specified, the problem of the different byte order is avoided. Furthermore, Java provides the mechanism to send programs from one computing node to the other.

The additions to the native language are similar to those introduced to task parallel programming languages like FortranM or CC++[41, 16, 18]. Thus, it is possible to generate, with an appropriate translator, a FortranM or CC++ program for fine grain parallelism, as well as, a dataflow driven program for coarse grain parallelism. This is based on only one representation of the program. Hence, one can interpret the program flow of the enhanced data-process-graph in multiple program paradigms.

After the processes and the data objects 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

```
PROCESS huv (IN_DATA OBSERVATIONS,
IN_DATA MODEL_in, OUT_ DATA MODEL_out)

    !– Quality Control
    do i=1,NoOfObservations
    call buddy_check
    call gross_check
    end do
    !– Matrix Solve
    do i=1,NoOfGridpoints
    call set_up_the_matrix
    call solve_the_equations
    end do


END PROCESS huv
```

Figure 8.9: *Definition of a process object using data objects on its inputs.*

be compiled on one machine and is not portable), may limit the number of choices for the mapping. To minimize the overall wall-clock time of the program execution, dynamic load balancing is 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 8.4 shows an example of some system variables monitored to support the mapping strategy. Here the CPU Load, Load Average, and Swap Load are displayed. The load monitor helps to display performance bottlenecks of the parallel program during its execution on the machines, while collecting a time space diagram. The user of the metacomputer might find it more useful to incorporate the state of the computation in a performance window, as shown in Figure 8.7. The active processes at a given time are displayed with a circle around them. To give information about possible congestion, the messages in a message queue for one process module are added with small circles on the arc between producer and consumer.

In the example depicted, all processes are mapped to an SP2 and the graphical output is

viewed on a SPARC workstation. In case the processes are written in a portable way and compilation on other machines is possible, different process-machine mappings could be used. The dynamical execution of a program is driven by two factors:

1. The software modules available for the different hardware platforms.
2. The availability of a computational resource.
3. The utilization of the computational resources.

Figure 8.10 illustrates the process responsible for making the selection of the hardware and software used to execute the program on existing hardware platforms.
In many scientific programs, a problem is solved many times for similar instances of data. They do not lead to substantial differences in the execution time. The information about the execution time on different machines is stored in a database. This can either be achieved via direct measurement, or a performance prediction analysis algorithm. Once the suspected execution time for a particular machine configuration is stored in the database, the information about the current utilization of the machine is used to predict the real time performance of the program. In case several choices of software and hardware mappings are available, the one with the shortest execution time is chosen. Hence, the selection not only includes a hardware mapping, but can also include the usage of completely different algorithms to solve the (meta)problem, best suited for the selected computer.

Internal Structure

Figure 8.11 shows the multiple purpose of the metacomputing editor. As mentioned before, the editor is used to simplify generating parallel programs or transferring a sequential program into a parallel. Additionally, it can be used to supervise the execution of a sequential, as well as, a parallel program.
Internally, several layers exist to interface with low level message passing routines. For the current implementation, a port to MPI is under development. While providing interfaces to different languages and message passing libraries, a wide variety of software support can be granted. On top of the communication library, a *Parallel Support Library* provides the necessary functions, to simplify parallel programming for distributed vectors. The parallel vector routines are necessary for the NASA DAS and can be reused by other similar scientific

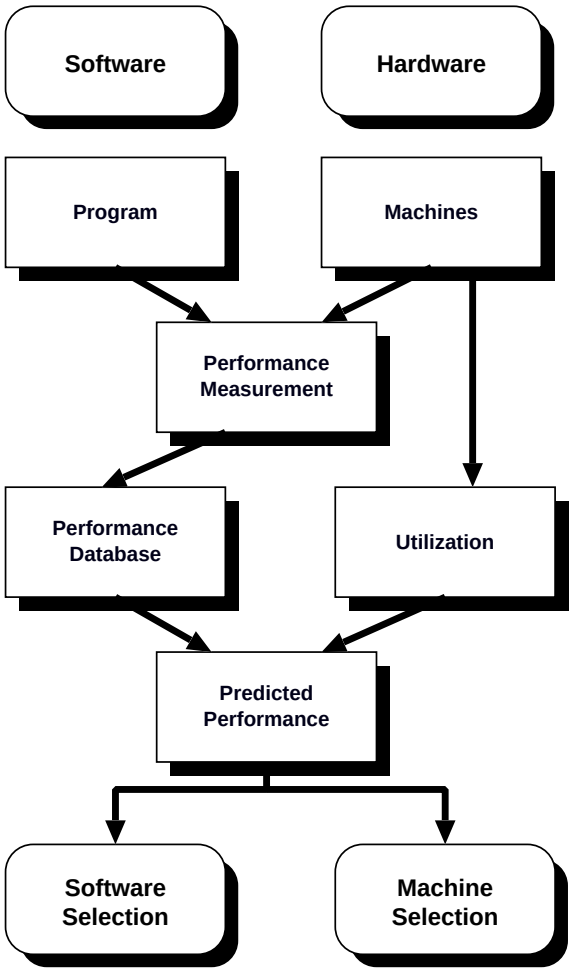

Figure 8.10: *Dynamical selection process during program execution*

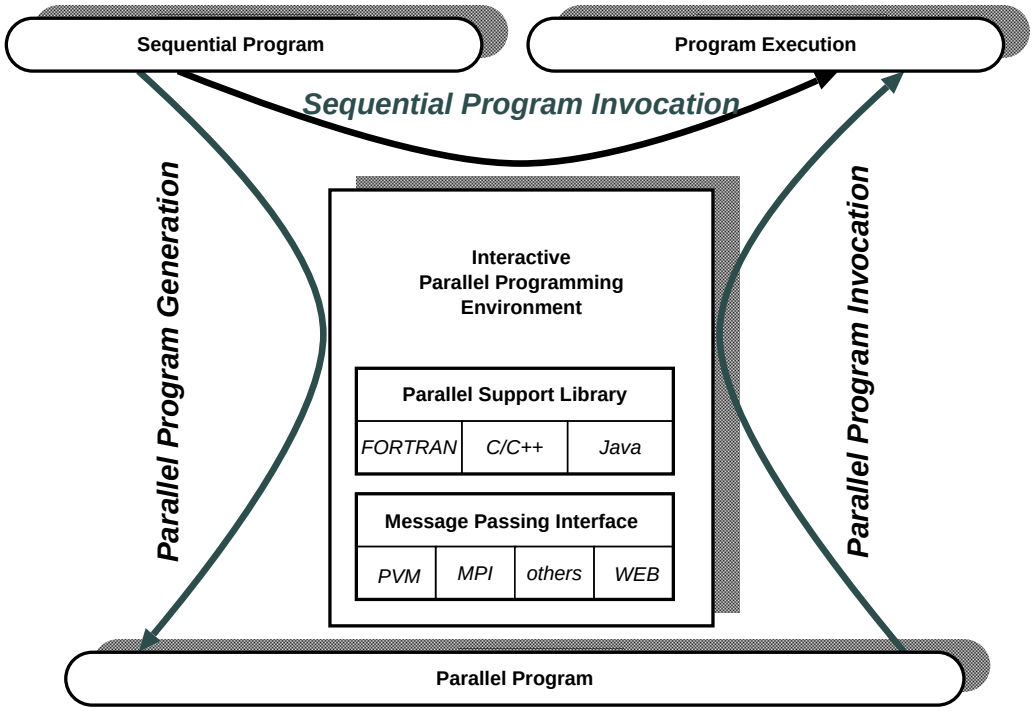

Figure 8.11: *The multiple purpose of the parallel programming environment while creating and executing parallel programs.*

codes[119]. It is possible to integrate other standard libraries, like ScaLAPACK[20], to enhance the function of the middle layer of the metacomputing editor.

While the original prototype of the metacomputing editor was developed in Tcl/Tk, Perl, CGI and Python, the current implementation of the interface, is based on Java[87, 127, 111, 77, 38]. This simplifies the expansion towards the WWW driven usage of the interface, as suggested in the MetaWeb project[17]. Using Java also reduces the number of software packages involved in the core implementation of the computing environment, thus making the environment better maintainable.

Even though Java is supposed to be platform independent, the usage of threads is currently not. On different platforms, preemptive or non-preemptive scheduling is used. Writing programs with a non-preemptive scheduling strategy is easier. It would be desirable for Java to provide a method switching between the scheduling policies, so that one can establish a uniform program for all platforms supporting Java. Furthermore, a serious bug while

launching runtime processes exists in several versions of Java, thus preventing reliable usage on some platforms[69]. We expect that the current version of Java (1.0.2) will be changed to overcome these problems.
To demonstrate the flexibility of the data flow concept and visual programming, another important utilization of the metacomputing editor is introduced in the next section.

8.1.2 A Loosely Coupled Metacomputer

The previous example was based on the medium grained problem for the NASA DAS. The resulting program had to be executed with the highest possible speed, thus the environment was used to produce a message passing parallel program based on MPI. The program is then executed on a supercomputer in stand alone fashion. No corporation is necessary between supercomputers, other than forwarding the result to the workstation where the graphical display is performed. The resulting message passing program is usually run in batch operation on the supercomputer, thus interactive program control is not available. A method has to be derived to incorporate computing resources operating in batch and in time sharing mode.
We classify typical usage of a supercomputer in grand challenge applications in three groups:

1. Executing large massively parallel jobs in batch operation.
2. Executing large massively parallel jobs in time sharing.
3. Executing a number of jobs built from the first two categories.

Figures 8.12- 8.15 show an example of a program designed for a loosely coupled metacomputer. The task of the program is to use some input data, process it with a program, forward the output to another program and show the result of the overall calculation on a terminal (Figure 8.12). Program A is mapped onto a DEC Alpha workstation farm, while program B can be alternatively mapped onto the SP2's from NPAC, Cornell, and Maui. A uniform job description form, as displayed in Figure 8.15, is used to specify differences for each machine due to the file system and user accounts, as well as, the differences in the batch program operating at the different sites. For machines running in time sharing mode, it is not necessary to fill out such a form (NPAC SP2). After the batch jobs have been properly defined, the program can be executed. A queue list is used to supervise the running jobs on the different

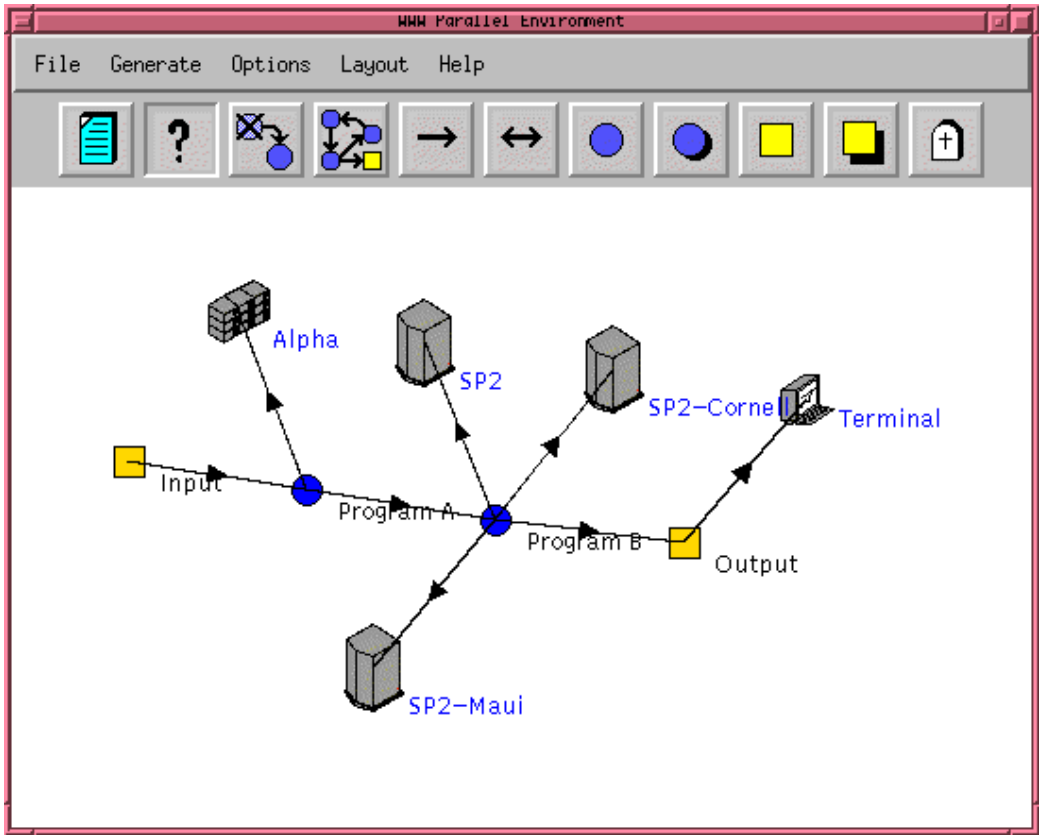

Figure 8.12: *A loosely coupled metacomputer program.*

machines (Figure 8.14). Each job is assigned a unique identification and can have input and output data dependencies with other jobs, (e.g. a job has to wait for the completion of the jobs given in the input dependency list, before it can be started). Once a job is completed, the state is updated in the job list and dependencies with other jobs are resolved. Jobs which do not depend on any other jobs (their dependencies are resolved), are then submitted for execution. Once the job is running on the particular machine, the queue list is updated again. The selection of jobs and their execution is repeated, until all dependencies are resolved. It is a straight forward implementation of a classical dynamical dataflow concept using global control.

Internally, the dataflow graph is converted to a sequence of shell commands which are executed in a special Java thread. The thread is responsible for scheduling and supervising the

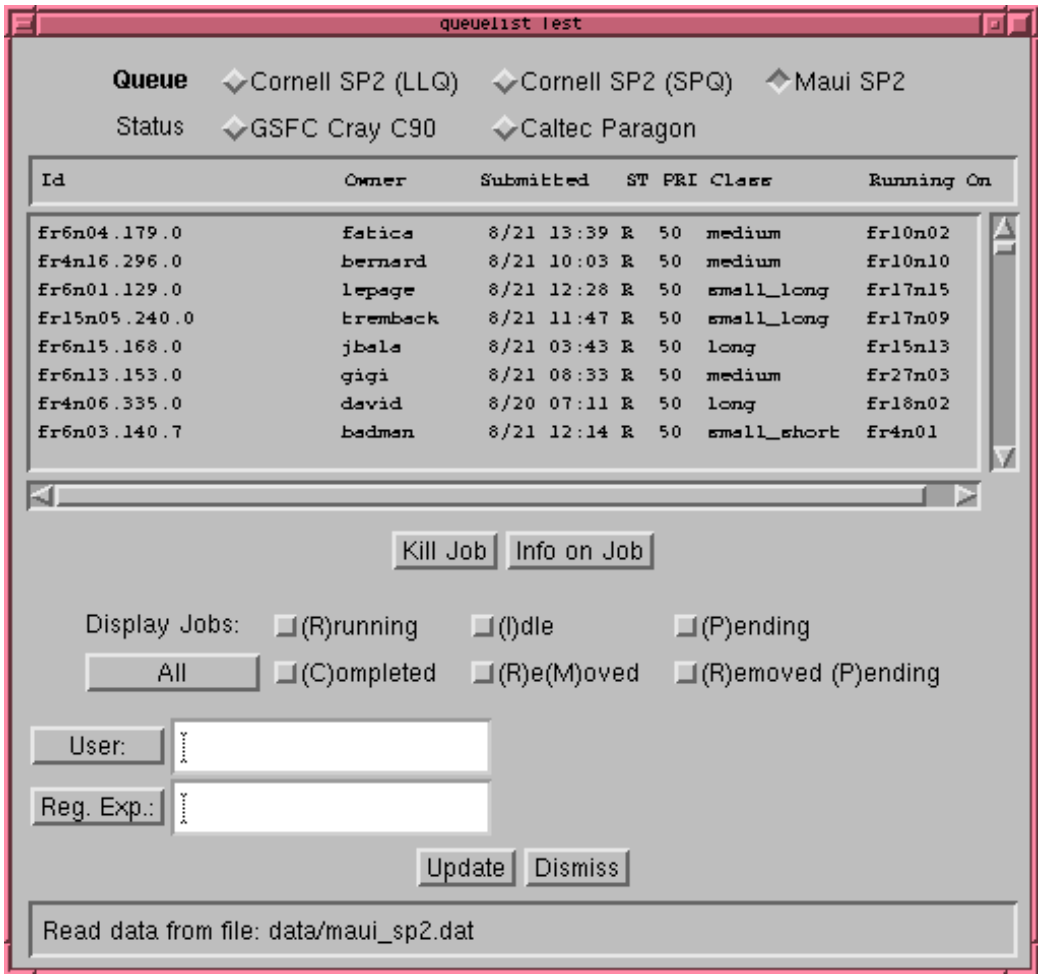

Figure 8.13: *The list of jobs submitted to the supercomputer.*

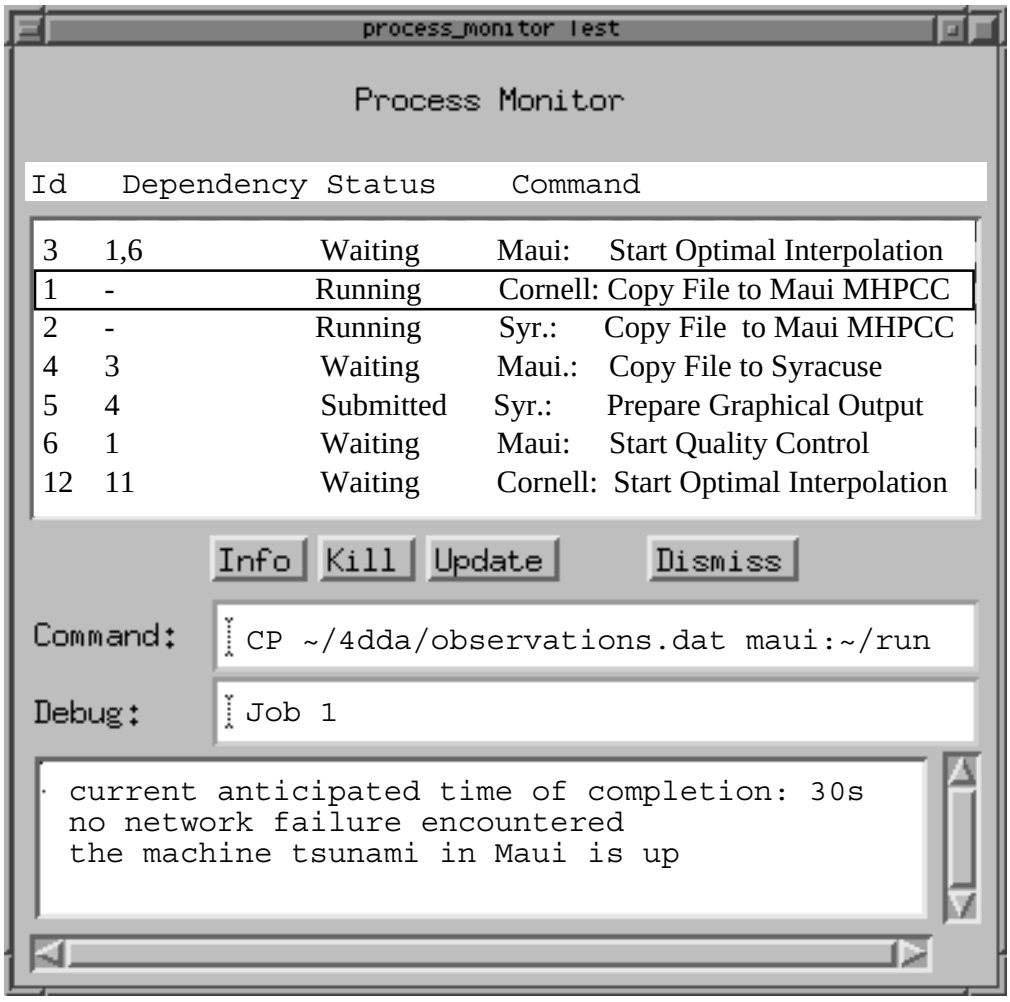

Figure 8.14: *The list of jobs submitted to the metacomputing environment.*

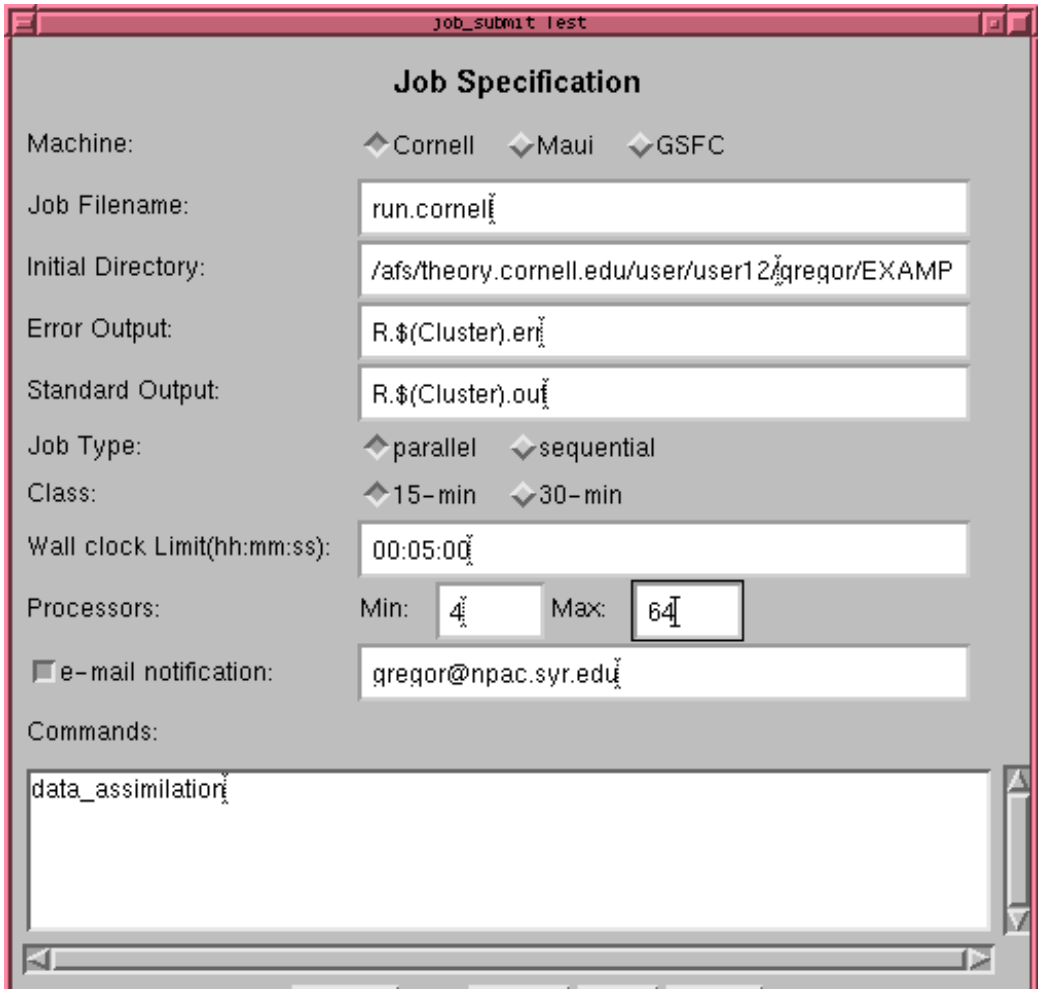

Figure 8.15: *The job submission form for supercomputers operating in batch mode.*

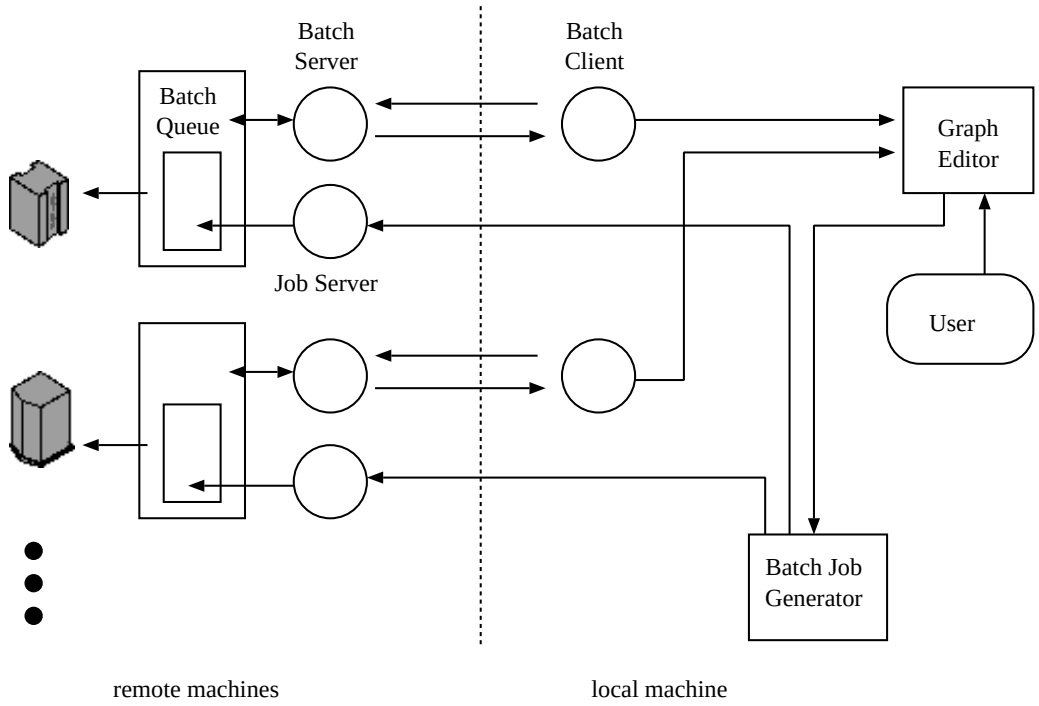

Figure 8.16: *The details of the loosely coupled metacomputer.*

parallel jobs which are executed asynchronously. For each job, a separate thread is created. A job can be in the following states:

Submitted – the job is submitted and prepared for running on the target machine.

Running – the job is running on the target machine.

Completed – the job is completed and the result is available for other jobs.

Failed – the execution of the job has failed.

Waiting – the job waits on the completion of one of its dependencies (input resolution).

Halted – the job has been halted by the operator.

A separate list is used to collect failed jobs. They are either resubmitted, or after a particular timeout, forwarded to another component of the metacomputing environment. Global control is used to supervise the jobs in the batch queue.

Job Replication

The mapping of jobs to machines operating in time-sharing and batch mode, forces one to use different strategies for the prediction of the runtime of a job. In time sharing mode, the system values, like current load and load average, are used to predict the completion time of the job. This is especially useful when a performance prediction function is available for the job.

In case the performance prediction is not possible or the job is executed on a supercomputer operating in batch mode, the execution time is not as easy to predict. One way to solve this problem is to use a global queue and to submit all jobs to the global queue. The decision regarding on which machine is executed, is performed by this queue. The batch operating software CODINE is able to do this[22, 6]. If this queuing system is not installed for the machine, as it is in the current setup of the metacomputer, the situation is more complex.

Thus, the jobs are simply replicated on each one of the machines. The first job completed causes the other jobs to be terminated. They are removed from the queue. The disadvantage of this strategy is that in case jobs are started at the same time, expensive resources in CPU cycles are wasted. Thus, one has to ensure to kill the jobs on all but one machine. The machine on which the job is left is selected via its performance characteristics.

The supervision of the status of the machines can be achieved in multiple ways. It is possible to run a Java server on the supercomputer side and send information about the status upon request. The current setup allows the user to start a remote procedure call and look into the job queue. Information about the status of the running jobs are obtained from the remote procedure call. Figure 8.16 depicts the details of the loosely coupled metacomputer.

Practical experience has been gathered while executing large jobs using more than 100 processors on the SP2 in Cornell and in Maui. During these experiments the jobs have been submitted to both machines. Whenever a job got accepted in a machine, the corresponding job on the other machine got deleted. With this simple strategy, a reduction of 3-6 hours wall clocktime for the completion of the job could be achieved. Therefore, the turnaround time of the computations to be performed is drastically reduced during peek hours.

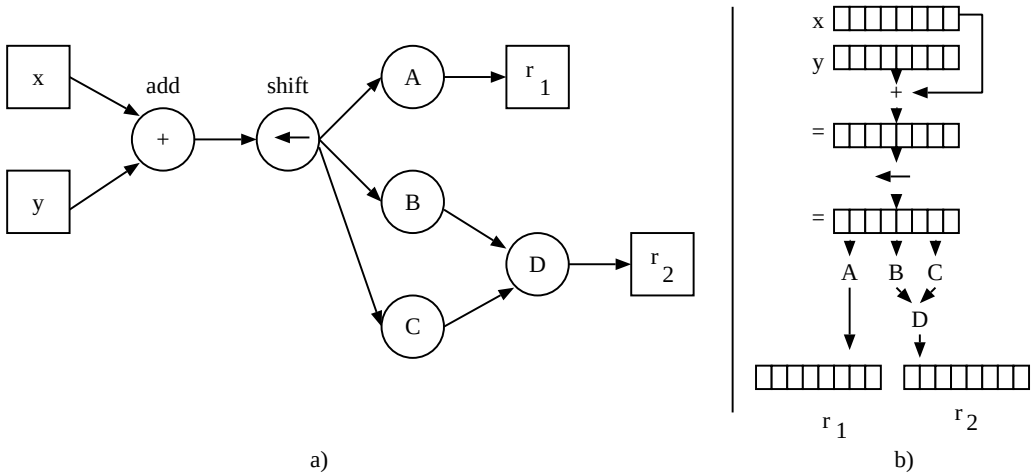

Figure 8.17: *Different paradigms expressed in a dataflow graph*

8.2 Dynamic Resource Management

This section illustrates the function of the dynamic resource management on an elaborate example.

Figure 8.17 and 8.18 show the problem to be solved. Here, a program is displayed which adds first two vectors, x and y. Then, it performs a shift and the the procedures A, B, and C are started using the result of the shift operation. The results of the calculation B and C are forwarded to D and the final result r_1, and r_2 are obtained. A possible parallel execution is depicted in Figure 8.18. A CM5, a SP2, and a workstation are utilized.

Step 1 – The resource mappings are specified and the program is executed.

Step2 – The data objects are mapped onto the CM5. A request is issued to test if the SP2 or the CM5 have enough computational resources to perform the addition of the two vectors. Let us assume, the CM5 has enough resources.

Step3 – Then, the addition is performed on the CM5 and new requests are issued to test if the shift operation can be performed on either machine. Let us assume, the CM5 has enough resources.

Step4 – Then, the shift operation is performed on the CM5.

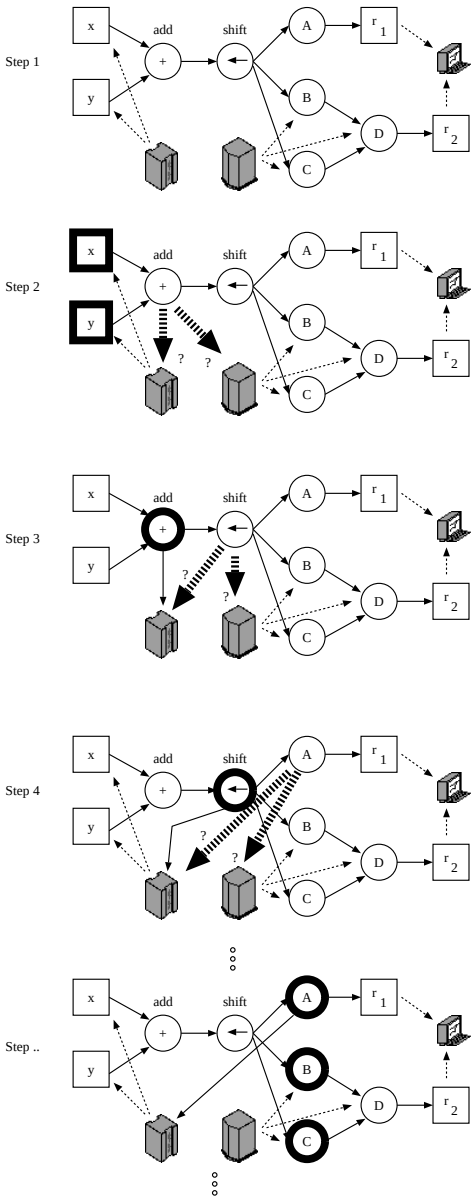

Figure 8.18: *Different paradigms expressed in a dataflow graph*

This is iterated for the whole computation. The last step in Figure 8.18 depicts a state where multiple processes can be active. Process A is mapped onto the CM5 and occupies its resources. Thus, the processes B and C are mapped onto the SP2, after the result of the shift operation has been transported to the machine. A truly heterogeneous execution of the programs is performed. Because the processes have been executed on both machines, it is advantageous to compile the instructions of the process, for both hardware platforms, in advance and store them in a library, locally accessible for the machine. Thus, besides the decision on which machine the process is executed, the decision about which software is used, is implied by the hardware choice.
The programs generated in this way cause some overhead by using message passing to communicate with each other. In case the mapping of a sequence of processes is forced on only one machine, a sequential program is generated instead of a message passing program. This is essential for the use of the program environment for grand challenge problems, where speed is a major factor.

8.3 The Metacomputing Library

Two libraries for users and developers are distinguished. Their function is similar, but their access mechanism is bound by different priorities of users and developers. Both libraries store

1. the module source,
2. the machine on which the module/program can be compiled,
3. the compilation option for each machine,
4. a runtime prediction function, which depends on the number of processors used and on the values of the input parameters to the module (usually referred to as the problem size),
5. a table noting the actual runtime of the module during a previous run for each machine (if available).

The values in the tables are normed, based on the exclusive usage of the machine. In addition, there is a temporary library available in each computational unit which is attached to the metacomputer. This library can hold precompiled modules. The timing information stored in the library is used by the resource management environment to schedule the module execution. It is expected that the available libraries will grow rapidly and an information exchange is implicitly achieved while using the metacomputing environment. This can be of advantage for industrial usage, where predefined solutions are often acquired[49] through rapid prototyping. The idea for such a library evolved from the extensive performance measurements for solving a system of linear equations for LU factorization with different strategies in a parallel computer using different numbers of processors. Information similar to those published in [126, 124], can be used to make the decision about which algorithm and how many processors, as well as, which platform have to be used to minimize the execution time.

8.4 The Metacomputer Resource Monitor

Because of the increased number of resources included in the metacomputing environment, a substantial amount of research should be spent on the issues involving the supervision of the metacomputing environment by an operator. This includes security, and resource management.

Because there are so many resources combined in a metacomputer, and many users are expected to use the environment, a central control seems to be impractical. This is also due to the fact that currently, the supercomputing centers do run separate batch queues.

To avoid congestion while monitoring the available resources in the metacomputer, a hierarchical approach is appropriate. Figure 8.20 displays the view of such an hierarchically organized metacomputer.

On the lowest level are the compute servers. They are connected to resource servers, which supervise the resources available in a small compute cluster. Resource servers are connected to other resource servers, which in turn, reflect the resources available in the clusters associated with the resource server. Resource allocation of the machine is first done in clusters close to each other to reduce network traffic. Thus, it will minimize intra-cluster communication. For the practical operation of the metacomputer, it is important that resources can be dynamically added and removed. Because of this dynamical behavior, fault tolerance has to

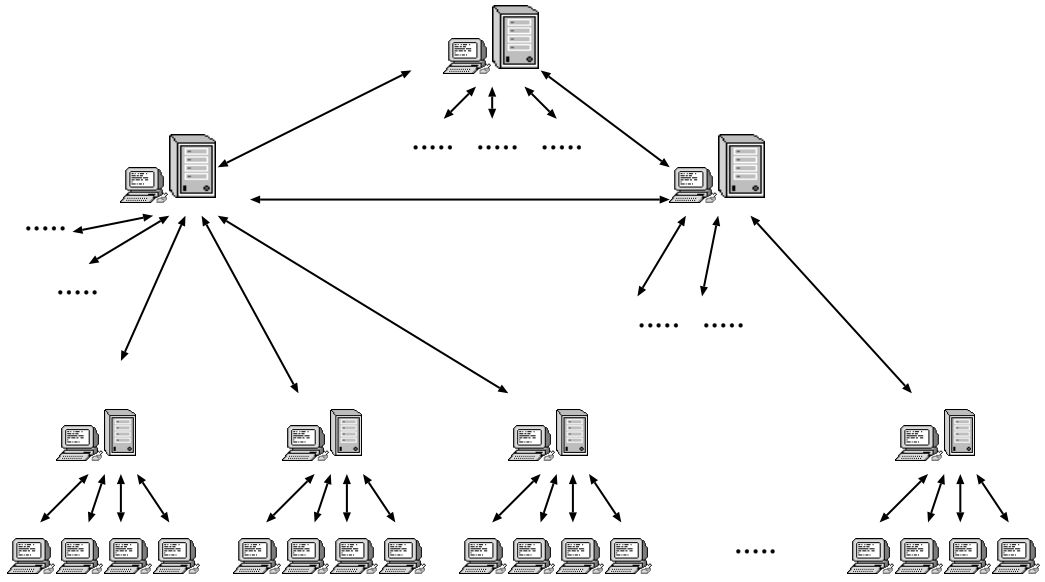

Figure 8.19: *The WWW Metacomputer.*

be maintained. This involves the remapping of a job to another component of the metacomputer in case the original component fails, or is removed from the metacomputer. Job runtime limits have to be established in order to prevent blocking of resources by one job. Load information and queuing status of the different components can be activated with the geographical map, similar to the one shown in Figure 7.6.

8.5 On Demand Publishing

Frequently, the data produced for grand challenge problems are too large to store. The reproduction of the experiments are a preferred alternative. Since it is expected that the users of the data are non experts in parallel programming, even the parallel programming environment is too complicated. The scientists are not interested in the actual execution of the program. Their requirements are:

1. Easy access,

2. Correctness,

3. Availability of the result.

On demand publishing provides the solution to this requirements. Today, WWW browsers are familiar to a wide computer user community[50]. Thus, it is natural to incorporate the access to the program solving the grand challenge via a HTML page. This page can be publicized with the help of an accessible HTTP server. Methods are included in the page to start the program, and display the result.
For the NASA assimilation system, it is most natural to give the scientist some control of the input parameters. This is shown in Figure 8.20, where various parameters can be modified. Normally, the parameters would have to be changed in the complex original FORTRAN program and data files. Changing the parameters requires recompilation. Because of the modular properties of the 4DDAS, it is advantageous to design the page similar to the module flowchart, as introduced in Figure 2.5. Then, the resulting output can be distributed in an appropriate form, e.g. GIFs accessible through additional WEB pages or a down-loadable file including the datasets in a predefined format.
Besides the reproduction of on demand data, the NASA project provides a set of standard experiments useful for many researches in the atmospheric science community. In this case, it is a waste of resources to reproduce the dataset over, and over again. Thus, it is sufficient to store the data set, and the programs interpreting and displaying the dataset, on an FTP server. Download can be activated via anonymous FTP, incorporated in an HTML page, which is accessible on the World Wide Web(WWW). A step in this direction for the NASA DAS is the planned Data Online-Monitoring System (DOLMS) by DAO[27]. Here, the selection of single GIF images are used to distribute results.
The amount of data involved for displaying, requires a fast data visualization package. Providing a WWW interface to a sequence of downloadable GIF pictures will not provide the speed required. Thus, it is better to download the considerably large dataset directly to the local machine to do further analysis and display offline. Another way, is to use Java scripts and forward the GIF images, in a pipelined fashion to the requesting machine, even before the figure is actually demanded.

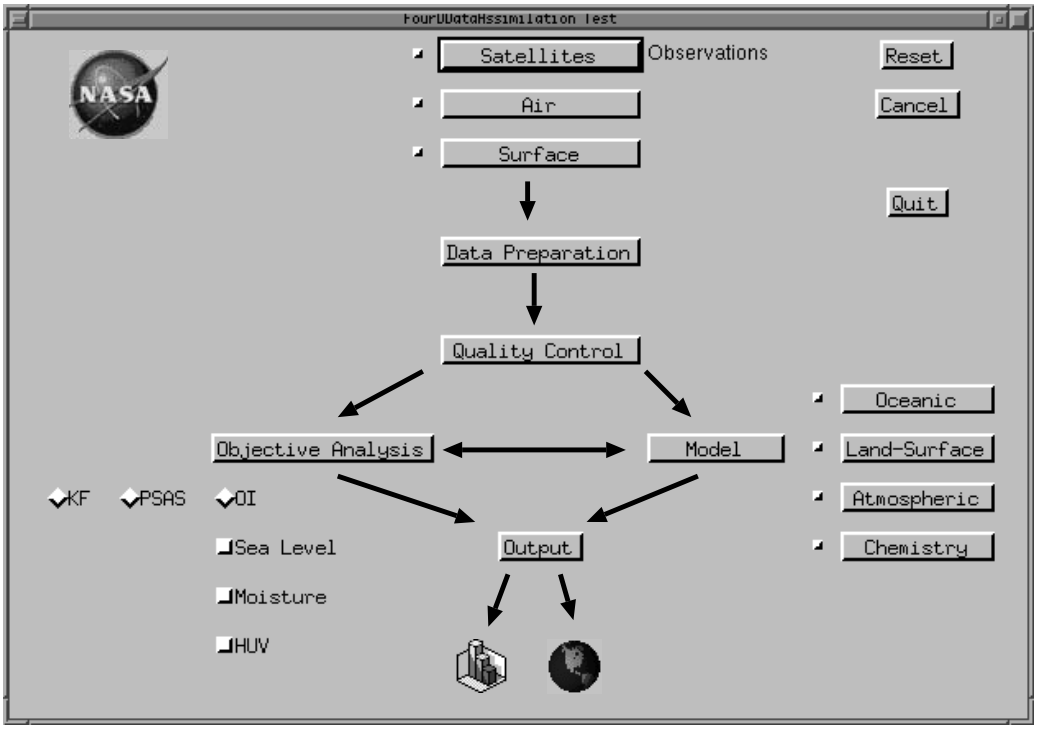

Figure 8.20: *A WWW interface for the on Demand calculation of the 4DDAS.*

8.6 Dataflow, a Multiparadigm Program Notation

This section answers the question:

Can dataflow be used to express programs in a multiparadigm way?

The answer is yes, and from the considerable amount of theoretical work spent in the definition of dataflow languages to be expected.

Program Notation

As seen before, the dataflow specification of the program has to be translated into an appropriate parallel program. Depending on the resources and the programming paradigm different translations are possible. They can be executed on different machines, even though, the program is only specified once. To show the usefulness of this multiparadigm programming concept, it is referred to Figure 8.17 a) and the Programs 8.1-8.6. The original program can be translated in an arbitrary parallel or sequential programming language.

Program 8.1 An example formulation in a CSP like program.

```
1  Vector x, y, w, r1, r2, t1, t2;
2  do sequential
3      w ← VectorShift(VectorAdd(x, y), left);
4  end
5  do parallel
6      r1 ← A(w);
7      do sequential
8          do parallel
9              t1 ← B(w);
10             t2 ← C(w);
11         end
12         r2 ← D(t1, t2);
13     end
14 end
```

The first intuitive formulation of a CSP[63] like code (Program 8.1) shows that the parallel program is more complicated than the visual representation makes believe. Nested statements are used and temporary variables have to be explicitly used to express the parallelism.

Using a message passing paradigm, makes the program even more complicated, as shown in Program 8.2. Writing the program in a dataflow language makes the program notation even more difficult because firing rules have to be established[61, 86, 14]. The power of dataflow computation becomes more obvious if a language is defined which implicitly generates the firing rules. Program 8.3 shows the notation in a language using such implicit semantic rules. Nevertheless, now it is still more complicated to see the the dependencies between the separate calculations. In case the program is more complex, the problem increases. Therefore, the graph representation is still preferable. Other program representation can be immediately derived from the process graph. Programs 8.5-8.6 show a sequential Fortran77 and Fortran90 program, as well as, an HPF program.

Dataflow Engine

Many ways to enable the dataflow concepts on distributed computers exist.
In the first approach, each computational unit acts as a *dataflow processor*, as known from the hardware implementations used in image processing. Each processing unit listens to a port and grabs the data which is necessary for a computation stored in the computational unit. Once all the data for a calculation is agglomerated, the command is executed and new data is generated. The new data is then distributed to the network and flows between the processing units, as long as it is not used by another command. It is essential to include special replication of the data in case it is used by several commands. Thus, each data has an additional tag connected to it representing the number of times it has to be picked up by a processing unit before it can expire. This type of dataflow is useful when few processors are used, and the message and communication overhead is small, as it is in certain applications of image processing. In case it is not desirable to use this truly distributed concept of dataflow, an easier implementation can be achieved with the help of a global scheduling mechanism and policy. In the current implementation of the metacomputing environment, this strategy is used. Here, a global processor supervises the execution of jobs and the progress of the program is monitored during each time step. This approach is especially useful for coarse grain parallel programs. Other methods are modifications to the ones described before.
It is interesting to simulate a *resource driven* dataflow. Here, data is kept in the computational unit, as long as there is no unit found where the computation on the data can be performed. In fact, this additional feature is implemented in the metacomputing environment.

Dataflow Representation

A nice feature of the metacomputing editor is the ability to change the appearance of the processing and data nodes. One can define icons which can then be associated with the different nodes. In case a programmer prefers to visualize the vector structure of the problem, vector icons could be used. An example for the diversity of the display, is shown in Figure 8.17 b). The flow of data has not changed.

8.7 Related Research

A lot of research has been done in the field of visual programming, in respect to parallel computing.
The environment introduced in this paper, uses similar approaches as, i.e., HeNCE and CODE. Nevertheless, it extends the usage towards a realistic metacomputer while providing a database of performance predictions, which guides the selection and mapping of programming tasks to selectable resources.
A more detailed description of visual programming and their applications in parallel computing can be found in literature[14, 86].
A short summary of tools which will have an impact on the further improvement of the metacomputing environment are listed next. Tools are distinguished which are used for visual programming, as well as, metacomputing.

8.7.1 Visual Programming

CODE

CODE is a visual parallel programming system, which allows users to compose sequential programs into parallel programs. The parallel program is displayed as a directed graph, where data flows on arcs connecting the nodes representing the sequential programs. The sequential programs may be written in any language. CODE will produce parallel programs for a variety of architectures because its model is architecture-independent. The CODE system can produce parallel programs for machines running PVM, as well as, for the Sequent Symmetry.

HeNCE

HeNCE is a metacomputing environment based on PVM[61]. Its goal is to greatly simplify the software development cycle. It implements a system for source file distribution and compilation on remote nodes. Source files can be compiled in parallel on several machines. This task is controlled by a global process manager. HeNCE lacks virtual machine management facilities, since its primary goal is to simplify the programming task.

AVS

The Abstract Visualization System (AVS) is a commercial visualization environment available on a wide variety of compute platforms[5, 110] . It is based on the software engineering definition of dataflow. A flow network of autonomous processes is generated visually. Data is passed between input and output ports of the processes connected via edges. Each process module fires as soon as its inputs are present. Thus, it reflects a statical dataflow model. AVS provides a convenient type checking on its ports. Message passing occurs only through the input and output ports. A global control manager coordinates the data transfer mechanism which is internally carried out by an Interprocess Communication(single machine) or Remote Procedure Call mechanism (distributed machine). Other similar systems to AVS, are Explorer from SGI, Data Explorer from IBM, and Khoros [73, 97, 24].

8.7.2 Metacomputing

Legion

The Legion project is pursued at University of Virginia[76]. It attempts to create system services for wide-area assemblies of workstations and supercomputers which provide the illusion of a single machine. The Legion system is an object-oriented system based on C++. The Goal of the Legion system is to provide a shared object and shared namespace while allowing:

1. parallel processing,
2. resource management,
3. fault tolerance,

4. security,

5. wide-area network support, including managing facilities, and

6. improved response time for submitted jobs.

The central part of the Legion system, are legion objects which are located in a shared object space. The computational resources use this shared object space to exchange data with each other. Currently, a test-bed based on top of Mentat, a parallel C++,[109] is used for the implementation of a Campus Wide Virtual Computer(CWVC). The Goal of the test-bed is to demonstrate the usefulness of a Legion like system and to provide an interface to high performance distributed computing resources. A single unified file system is used to allow the easy access of files on different machines. A GUI allows the monitoring of the available resources which are accompanied by a resource accounting service. Automatic runtime scheduling is available. A debugger can support post mortem debugging.

WAMM

The Wide Area Metacomputing Manager (WAMM) is a project started in Italy involving several sites[128]. The Central part of WAMM is a GUI, which provides a geographical view of the system. Hosts are grouped in a tree structure following their geographical distribution. On the lower level are the hosts connected via a Local Area Network (LAN). Several LAN's can be combined to a Medium Area Network (MAN). All of them are connected via a Wide Area Network (WAN).
The WAMM internals are based on remote command execution of UNIX commands. The actions associated with the functions are displayed in the GUI. One aspect of the environment is the remote compilation of programs. Tools are provided, which greatly simplify this task. The program environment is based on PVM3, thus allowing the execution of parallel programs on previously selected machines.

WANE

The Wide Area Network Environment (WANE) is developed at the Supercomputer Computations Research Center. The design goals are high scalability, fault tolerance, and information encompassment. It is based on the integration of several ongoing software projects. WANE

provides a comprehensive internet service package, allowing servers to connect to the internet and make use of client software for several platforms. A WANE server provides extensive user access controls, hierarchical administrative levels, multiple user domains, and a user friendly GUI to add users to the environment. The client software provides several software packages for the internet connection, including WWW browsers, network connection packages, and many more. WANE is a simple way to get connected to the internet easily. Thus, it can provide a basis for the connectivity of resources to a metacomputing environment. Issues of programming in this heterogeneous environment are not addressed so far.

XPVM

XPVM provides a graphical console for PVM with support for virtual machine and process management[130]. The user is able to change the metacomputer configuration (by adding or removing nodes) and spawn tasks, in a way similar to WAMM. In contrast to WAMM, XPVM does not provide the same geographical view of the virtual machine. It is more suitable for smaller systems. XPVM does not include facilities for source file distribution, parallel compilation and execution of commands on remote nodes. It includes a section for trace data analysis and visualization, which is not yet implemented in WAMM.

Globus

The Globus project at Argonne National Laboratory, is developing a basic software infrastructure for the computations that integrate geographically distributed computational and information resources[42, 43]. The I-WAY project demonstrated several components of Globus. The focus of the Globus project is the development of low-level mechanisms that can be used to implement higher-level services. Furthermore, techniques that allow those services to observe and guide the execution are planned. Overall, a single low level-infrastructure is provided.

MetaWeb

The goal of the MetaWeb project at NPAC is to provide a scalable metacomputer management package, based on WWW technology. The project will build on existing knowledge and experience with the management of LAN based computing clusters. While establishing such

an environment, the through-put of user applications can be increased by utilizing networked computing resources. The environment will be truly heterogeneous, enabling the usage of PCs, MPPs and vector-supercomputers. The computer language Java is used to provide the services. Fault tolerance is a primary issue in MetaWeb. The status of a calculation is retained, even if the system reboots itself. In addition, application jobs are resumed or restarted on failure.
A first prototype has been used on the basis of Perl-CGI-scripts, C-modules and HTTP servers. It has been successfully used for the Rivest-Shamir-Adelman (RSA) Factoring challenge. It is a public-key cryptosystem for both encryption and authentication. Each party has two keys: the public key and a corresponding secret key. The secret key can be derived via factoring from the public key. The different calculations to obtain the factoring are embarrassingly parallel and can be distributed over a number of machines. The result is collected and the main answer is given. More information about the system can be found in [51].

8.8 Advantages and Problems with Metacomputers

The advantages of a functioning metacomputing environment are obvious. A metacomputing environment:

1. utilizes the strength of individual computers and programming paradigms,
2. distributes a computational problem over the available resources,
3. can dynamically modify the application,
4. provides transparency for the user.

Common problems associated with metacomputers are the:

1. improvement of the existent software,
2. improvement of the bandwidth of the networks,
3. improvement of the quality of the user interface,
4. software engineering complexity to design a metacomputer.

A metacomputer environment is not only useful for grand challenge applications, but also for national challenge applications and even smaller ones. A common example is the usage of distributed telemedicine, where different resources are utilized to support diagnosis, remote imaging, and expert system shells.

8.9 Current State and Future Research

In this chapter, the usefulness of an interactive parallel programming environment for scientific grand challenge problems were demonstrated. The environment makes it possible to view the available resources as a metacomputer and reduce the development time of parallel programs. Dynamic process assignment is used to assist the execution of the parallel programs on diverse computing resources. This not only includes the selection of the best suited hardware platform, but also the appropriate software for solving the problem.

Currently, the environment consists of many modules which have to be integrated with each other. Several aspects of the metacomputing environment need definite improvement. One of them is the metacomputing editor, which should allow hierarchical definition of modules in order to prevent cluttering of the display with too large graphs.

An independent research field is the performance prediction and utilization of the different machines. So far, only simple strategies are chosen which can be implemented in a reasonable time frame. A lot of research has to be done in the security issues of the metacomputing environment. The current implementation uses UNIX password authorization, before a job with the loosely coupled metacomputing environment is submitted or started. More complex security issues arise while distributing precompiled code with the help of world readable libraries. A method has to be established to prevent "Trojan horses."

Many other extensions are planned in future research projects. One of the most striking will be the inclusion of a message passing layer for the WWW. This will allow to use resources accessible via the WWW. Integration of a Fortran Interpreter, or language tools to simplify the distribution of programs, is desirable. For mathematical problems, scripting languages like matlab, scilab, or mathematica, could be viable alternatives to interpreted Fortran or Java.

Program 8.2 An example formulation in a message passing like program.

```
process main
  Vector x,y,w,r1,r2,t1,t2;
  w ← VectorShift(VectorAdd(x,y),left);
  sendto(w, A)
  sendto(w, B)
  sendto(w, C)

  r1 ← receivefrom(A)
  r2 ← receivefrom(D)

end process

process A
  r ← do calculation A
  sendto(r, main)
end process A

process B
  r ← do calculation B
  sendto(r, D)
end process B

process C
  r ← do calculation C
  sendto(r, B)
end process C

process D
 b ← receivefrom(B)
 c ← receivefrom(C)
 r ← do calculation D(b,c)
  sendto(r, B)
end process D
```

Program 8.3 An example formulation in a Dataflow like language with no program counters

$$
\begin{aligned}
&w \leftarrow \mathit{VectorShift}(\mathit{VectorAdd}(x, y), \mathit{left});\\
&r_1 \leftarrow A(w);\\
&r_2 \leftarrow D(B(w), C(w));
\end{aligned}
$$

Program 8.4 An example formulation in FORTRAN90 with program lines. The functions B and C are executed in the order determined by the compiler.

```
1 INTEGER X(N),Y(N),w(N),r1(N),r2(N)
2 w = EOSHIFT(x + y);
3 r1 = A(w);
4 r2 = D(B(w),C(w));
```

Program 8.5 An example formulation in FORTRAN77 with program lines. The functions B and C are executed in the order determined by the compiler.

```
1 INTEGER X(N),Y(N),w(N),r1(N),r2(N)
2 do i = 1,n
3     w(i) = x(i) + y(i)
4 end do
5 do i = 1,n − 1
6     w(i) = w(i + 1)
7 end do
8 r1 = A(w)
9 r2 = D(B(w),C(w))
```

Program 8.6 An example formulation in HPF2.0. In HPF 1.0 there is no easy way to incorporate task parallelism. HPF 2.0 will provide a special directive ON HOME.

```
1  INTEGER X(N),Y(N),w(N),r1(N),r2(N)
2  !HPF$DISTRIBUTE X(BLOCK),Y(BLOCK),w(BLOCK)
3          w = EOSSHIFT(x + y)
4          r1 = A(w)
5  !HPF$ON HOME p(1)
6          t1 = B(w)
7  !HPF$ON HOME p(2)
8          t2 = C(w)
9  !HPF$ON HOME p(3)
10         r2 = D(t1, t2)
11 !HPF$ON HOME p(4)
```

Chapter 9

Conclusion

In this dissertation, the following goals and results have been accomplished:

- It could be shown that the existent production version of the NASA data assimilation system can be parallelized for MIMD machines, besides vector supercomputers.
- Problems during the parallelization are encountered and solutions to solve them are found. Most of the problems were based on the original "dusty deck" approach.
- The sequential assimilation system could be optimized. Improvements have been forwarded to the originators, resulting in an improvement of the numerical accuracy of the code.
- A new deterministic quality control algorithm is outlined, which is planned to be incooperated into the next generation of assimilation systems at the NASA GSFC Data Assimilation Office.
- Different domain and functional decompositions have been analyzed.
- Different parallel algorithms based on the domain and functional decompositions have been implemented and tested. The analysis shows that a decomposition of the poles is necessary to avoid non-scalability of the algorithm.
- If more processors are used than the domain decomposition allows, the efficiency decreases.

- A simple algorithm has been derived using a hierarchical domain decomposition. The decomposition is based on block decompositions and cyclic decompositions, in horizontal and vertical levels. Its speedup is almost linear while using up to 100 processors.
- An analysis based on the simplification for parallel programming paradigms could be found. It has analogies to concepts as used in the field of atmospheric science. It is the well known dataflow concept.
- The design of a metacomputing environment is outlined, using the dataflow concept as basic programming paradigm.
- The dataflow concept can be used on different levels of parallel programming. This includes support for tightly coupled metacomputers and loosely coupled metacomputers. Programs running on tightly coupled machines use direct message passing between processes, while programs running on loosely coupled metacomputers use asynchronous message passing between "jobs".
- A dynamical dataflow model is preferable to support a loosely coupled metacomputing environment, in contrast to a static dataflow model.

9.1 Implications of the Grand Challenge Application on the Metacomputing Environment

The following list summarizes the implications of the grand challenge application on the metacomputing environment.

- Grand challenges deal with limited resources. A metacomputing environment must have an efficient resource management strategy to support grand challenge applications.
- Due to the complexity of the system, applications researchers are not able to learn everything. Most of the functions should be hidden. The user interface should be simple and intuitive. To reach acceptance it must be possible to integrate programs written in FORTRAN and other sequential programming languages into the framework. In the same way, other programming tools should be supported.

- A graphical metacomputing editor is useful to support program definition and execution.
- The distinction of data, process, and machine as nodes in a program graph is important. It can support mapping of data to a particular machine. The program flow is controlled with the help of edges in the set of data and process nodes.
- It is advantageous to support resource selection. This "violation" to the black-box metacomputing concept is necessary to achieve high performance, in case resource restrictions are known at compile time. This is done with a separate graph specifying the mapping between machines and data, as well as, processes.
- Different programming paradigms should be supported to use the architecture most suitable for the problem.
- Documentation support has to be granted to help improve the quality of the code. It is not useful to redistribute an undocumented code, nor to include it in a production version, because maintenance will become too expensive in the future.
- It is desirable to support fault-tolerance and automatic process or job migration in order to minimize the overhead for the user, in case problems occur during the program execution.
- To reuse components by others, a library of components should be supported. The library contains performance prediction data whenever possible. Once a function of the library is used and the performance data is useful for reusability, it is stored for future predictions.
- Batch operation should be supported, to allow the submission of large repetitive jobs.
- Interactive operation should be supported, to allow easy code development.

9.2 Future Avenues of Research

The following future research topics are revealed by the work conducted:

- The complete redesign of the 69,000 line OI algorithm is desirable. The data replication, as used in the current model variables, can be overcome while developing a localized data assimilation algorithm. The algorithm should be based on the definition of a local grid, rather than the availability of a globe grid representation.
- Though one of the goals of Java is to be platform independent, it has different thread scheduling policies on different hosts. Thus, program development with threads is more complex than it should be. It is desirable to extend Java to incorporate a switch controlling the thread scheduling policy. Preemptive and non-preemptive scheduling should be possible.
- A bug in the runtime environment of Java prevents the usage of the runtime method on some platforms. An alternative is to rewrite the runtime environment or provide the execution of operating system calls in a separate C or C++ application, which communicates with the Java applet.
- Many modules of the metacomputing environment have been prototyped. The incorporation and more thorough, testing should be performed.
- The issue of security has to be studied more thoroughly.
- The development of a reliable and fault-tolerant WWW message passing library is of utmost importance to integrate computational resources available on the WWW.

Due to its economical feasibility, the utilization of resources on the WWW can help solving grand challenge problems.

Appendix A

Abbreviations

Abbreviation	Meaning
CO_2	Carbondioxide
N_g	be the number of observations, effecting a particular grid point g
$\mathbf{A_g}$	be the resulting analysis at grid point g
$\mathbf{F_g}$	be the first guess value at the grid point g
$\mathbf{F_i}$	be the first guess value for the i^{th} observation
$\mathbf{O_i}$	be the i^{th} observed value
$\mathbf{W_{gi}}$	be the yet undetermined weight function
4DDA	Four Dimensional Data Assimilation
4DDAS	Four Dimensional Data Assimilation System
AS	Assimilation System
AVS	
CPU	Central Processing Unit
DAO	Data Assimilation Office
DAS	Data Assimilation System
ECMWF	European Centre for Medium Weather Forecast
FLOP	MFLOPS, GFLOPS, ...
GCM	General Circulation Model
GSFC	Goddard Space Flight Center
GUI	Graphical User Interface
HPF	High Performance Fortran
HUV	Height-u-wind-v-wind (analysis)
HiPPI	
IEEE	
MIMD	Multiple Instruction Multiple Data
MIX	Moisture-vapor (analysis)
MPI	Message Passing Interface
MPP	Massively Parallel Processors
NASA	National Aero and Space Agency
NPAC	Northeast Parallel Architectures Center
NQS	Network Queuing System

Abbreviation	Meaning
OI	Optimal Interpolation
OOMPI	Object Oriented MPI
PC	Personal Computer
PSAS	Physical space System Assimilation System
PVM	Parallel Virtual Machine
SIMD	Single Instruction Multiple Data
SLP	Surface-level-pressure (analysis)
WAMM	
WANE	
WORD	MWORD, GWORD, ...
WWW	World Wide Web
mb	millibar
s	seconds

Appendix B

Program and Code Examples

B.1 Generating Tightly Coupled Applications with the Metacomputing Environment

The metacomputing editor can automatically generate a message passing code based on Object Oriented MPI (OOMPI)[79]. Let, the following textual representation be the definition of a data object:

```
DATA observations BEGIN
  const int n=5;
  float x;
  float y;
  float temperature;
  float pressure;
  int model [n];
END
```

Then, the code for inclusion in the message passing program looks as follows:

```
// Author: Gregor Von Laszewski
// Tool  : GVL (Graphical Visual Language)
// Class : <TYPEobservations> derived from <observations>
< include files >

const int TYPEobservationsTAG = 201;
```

```
class TYPEobservations : virtual public OOMPI_User_type {
public:
  TYPEobservations(void) :
      OOMPI_User_type(type, this, TYPEobservationsTAG) {

    // The Datatype
    if (!type.Built()) {
      type.Struct_start(this);
      type << x ;
      type << y ;
      type << temperature ;
      type << pressure ;
      type.Entry(model,n);
      type.Struct_end();
      }
    };

  void Print(void) {
    cout << endl;
    cout << "Datatype: observations" << endl;
    cout << "float x : " << x << endl;
    cout << "float y : " << y << endl;
    cout << "float temperature : " << temperature << endl;
    cout << "float pressure : " << pressure << endl;
    cout << "int model[" << n << "] : ";
    gvl_Print(model,n);
  }

  void Set(void) {
    int rank = OOMPI_COMM_WORLD.Rank();
    Random(x,rank);
    Random(y,rank);
    Random(temperature,rank);
    Random(pressure,rank);
    Random(model,n,rank);
  }

private:
  // The Data for the class
    const int n=5 ;
```

```
    float x ;
    float y ;
    float temperature ;
    float pressure ;
    int model [n] ;
  // static variable to hold the new datatype
   static OOMPI_Datatype type;
};
OOMPI_Datatype TYPEobservations::type;
```

A simple application for the routines generated form the data definition block is given below. It emulates a ring application:

```
int main(int argc, char *argv[]) {
  OOMPI_COMM_WORLD.Init(argc, argv);
  int rank = OOMPI_COMM_WORLD.Rank();
  int size = OOMPI_COMM_WORLD.Size();
  TYPEobservations msg;
  if (rank == 0) {
    for (int i = 1; i < size; i++) {
      OOMPI_COMM_WORLD[i].Recv(msg);
      cout << " from " << i << " : " ;
      msg.Print();
    }
  } else {
    msg.Set();
    OOMPI_COMM_WORLD[0].Send(msg);
  }
  OOMPI_COMM_WORLD.Finalize();
}
```

Internally, a Java program generates the program codes as listed previously. The Java class has the following methods:

```
class DataToMPI {
```

```
  < global variables >
  < some other private functions >
```

```
  private void print_oompi_program () {
        -- prints the oompi program
  }
```

```
  private void read(String filename) { reads the file into the memory}
```

```
  private void ParseFile() {
        -- parses the file and extracts the variables as
        --    well as their types and dimensions
  }
```

```
  public void generate_oompi(String filename) {
        -- generates an OOMPI code
        read (filename);
        ParseFile();
        print_oompi_program ();
  }
```

```
  public void generateTestRingCode() { -- generates a test program }
```

```
}
```

B.2 Generating loosely Coupled Applications with the Metacomputing Environment

Once a loosely coupled job is generated with the help of the graph editor, it is transfered to an intermediate language, which is similar to a shell script. This is essential, because a programming of a tightly coupled metacomputer should also be allowed in textual form and not only in graphical form. Naturally, the graphical representation is preferred.
The language is built around a couple of simple commands allowing remote file access and remote compilation. Each command is attached with an *id*, which allows to generate dependencies between jobs. Jobs are all submitted asynchronously. A special *wait* command waits on the execution of the specified job. A *probe* command returns if the specified job has already returned. A special command is the *first* command, which returns as soon as one out of a specified list of jobs returns. The *cancel* command cancels a job on the appropriate machine.

B.2.1 Remote Script Language

A simple script for compiling and running a program calculating the heat equation using the SP2 at Cornell and Maui looks like the following:

```
MKDIR PROJ                                    id$mkdir
```

```
wait id$mkdir

CP heat/Makefile.maui"     cornell:PROJ/. id$1
CP heat/Makefile.cornell"  cornell:PROJ/. id$2
CP heat/cmd.cornell"       cornell:PROJ/. id$3
CP heat/cmd.maui"          cornell:PROJ/. id$4
CP heat/draw_heat.c"       cornell:PROJ/. id$5
CP heat/heat.h"            cornell:PROJ/. id$6
CP heat/make.mpi_heat2D.c" cornell:PROJ/. id$7
CP heat/make.ser_heat2D.c" cornell:PROJ/. id$8
CP heat/mpi_heat2D.c"      cornell:PROJ/. id$9
CP heat/ser_heat2D.c"      cornell:PROJ/. id$10

wait id$1-10

APPLY cornell:PROJ> make -f Makefile.cornell all id$make

wait id$make

jobno = SUBMIT cornell:cmd.cornell id$run

wait id$run

FINISH cornell jobno id$wait

wait id$wait

CP cornell:PROJ/final.dat . id$get

wait id$get
```

While using the concepts from task parallel languages, it is also desirable to provide a CSP lice notation. The OCCAM programming language might model here as a base for the concepts needed:

```
parallel (asynchronous) {
  cornell ::= sequential {
    MKDIR cornell:PROJ
```

```
    CP heat/Makefile.maui"     cornell:PROJ/.
    CP heat/Makefile.cornell"  cornell:PROJ/.
    CP heat/cmd.cornell"       cornell:PROJ/.
    CP heat/cmd.maui"          cornell:PROJ/.
    CP heat/draw_heat.c"       cornell:PROJ/.
    CP heat/heat.h"            cornell:PROJ/.
    CP heat/make.mpi_heat2D.c" cornell:PROJ/.
    CP heat/make.ser_heat2D.c" cornell:PROJ/.
    CP heat/mpi_heat2D.c"      cornell:PROJ/.
    CP heat/ser_heat2D.c"      cornell:PROJ/.
    APPLY cornell:PROJ> make -f Makefile.cornell all
    jobno_cornell = SUBMIT cornell:cmd.cornell
  }

  maui ::= sequential {
    MKDIR maui:PROJ
    CP heat/Makefile.maui"     maui:PROJ/.
    CP heat/Makefile.cornell"  maui:PROJ/.
    CP heat/cmd.cornell"       maui:PROJ/.
    CP heat/cmd.maui"          maui:PROJ/.
    CP heat/draw_heat.c"       maui:PROJ/.
    CP heat/heat.h"            maui:PROJ/.
    CP heat/make.mpi_heat2D.c" maui:PROJ/.
    CP heat/make.ser_heat2D.c" maui:PROJ/.
    CP heat/mpi_heat2D.c"      maui:PROJ/.
    CP heat/ser_heat2D.c"      maui:PROJ/.
    APPLY maui:PROJ> make -f Makefile.maui all
    jobno_maui = SUBMIT maui:cmd.cornell
  }
}

$job = first($maui,$cornell);

switch ($job) {
  case maui:     $machine="maui";    $kill="cornell"; $killid = $maui;    break;
  case cornell:  $machine="cornell"; $kill="maui";    $killid = $cornell; break;
}
KILL $kill $killid
CP $machine:PROJ/final.dat .
```

B.2.2 The Java Classes for Remote Computer Handling

The commands specified above are available as Java class.

```
public class META {

  public static int MKDIR (String machine,
                           String base_directory,
                           String directory)

  public static int RM (String machine,
                        String base_directory,
                        String file)

  public static int EXISTS (String machine,
                            String directory,
                            String file)

  public static void CP (String from, String to)

  public static void GET (String machine,
                          String directory,
                          String file)

  public static int APPLY (String machine,
                           String directory,
                           String runcommand)

  public static void getQueue (String machine,
                               String filename)
  < other routines >
}
```

B.2.3 Scheduling

Internally a program scheduler as described in the main part of the Dissertation, organizes the execution in a dynamical dataflow fashion.

```
class ProcessInfo {
  String  command;
  Process p;
  int     id;
```

```
  String  output;
  String  error;
  String  status;
  int     returncode;
  String  group;

  ProcessInfo(String status_msg)
  ProcessInfo()
}

class ProcessThread extends Thread {
  private static int    IdCounter    = 0;
  private static Array  process_list = new Array();
  private int Id;
```

```
  void synchronized SetId(int id)
```

```
  void synchronized Start()
```

```
  void Stop()
```

```
  void Suspend()
```

```
  void Resume()
}

class ProcessInfo {
  String  command;
  Process p;
  int     id;
  String  output;
  String  error;
  String  status;
  int     returncode;
  String  group;

  ProcessInfo(String status_msg)
  ProcessInfo()
}

public class SYSTEM {
```

```
  private static boolean verbose = true;
  static private int IdCounter = 0;
  static private Array  process_list = new Array();

  private final static int Pos_id      = 0;
  private final static int Pos_status  = 8;
  private final static int Pos_command = 20;

  public static void setVerbose(boolean verboseFlag)
  public static int RETURN_EXEC (String command, boolean echo)
  public static void BUFFERED_EXEC (String command, String filename)
  public static void RUN (String command)
  public static int RETURN_RUN (String command, boolean echo)
  public static synchronized void PrintRunningProcesses()
  public static synchronized void AddProcesses(List l)
  public static synchronized ProcessInfo Get(int id)
  public static synchronized String GetOutput(int id)
  public static synchronized String GetError(int id)
  public static synchronized void KillProcess(int id)
  public static synchronized void firstProcess(IntegerVector id)
  public static synchronized void wait(int id)
  public static synchronized void probe(int id)
  public static synchronized void suspend(int id)
  public static synchronized void restart(int id)
  public static synchronized void PrintOutput (int id, String filename)
  public static synchronized int oldEXEC (String command, boolean echo)
  private static void DEBUG(String s)
  private static void SEPARATOR()
  private static void ERROR(String s)
  public static synchronized int EXEC (String command, boolean echo)
  public static void emacs(String filename)
}

public class process {
  public static int    EXISTS (String file, String machine)
  public static void CP        (String file, String machine, String directory)
  public static void GET      (String file, String machine, String directory)
  public static void APPLY (String file, String machine, String directory)
  public static void getQueue (String machine)
  public static void getQueueMaui ()
  public static void getQueueCornell ()
```

```
}



public class JOB {
  public String IPName = new String();
  public String MachineName = new String();
  public String Filename = new String();
  public String InitialDir = new String();
  public String ErrorOutput = new String();
  public String StdOutput = new String();
  public String JobType = new String();
  public String Class = new String();
  public String Time = new String();
  public String Notify = new String();
  public int    MinProcessors = 1;
  public int    MaxProcessors = 1;
  public String Email = new String();
  public String Commands = new String();

  public JOB ()
  public void write_data()
  public void write_job_file (String filename)
}
```

Bibliography

[1] NCCS Science Highlights: Earth and Space Sciences, Supercomputing and Mass Storage Applications, 1994.

[2] Ahmad, I. *Dynamic Load Balancing for Large Distributes and Massively Parallel Multicomputer Systems.* PhD thesis, Computer Science Department at Syracuse University, June 1992.

[3] Akl, S. G. *The Design and Analysis of Parallel Algorithms.* Prentice Hall, New Jersy, 1989.

[4] Amdahl, G. M. Validity of the single processor approach to achieving large-scale computing capabilities. In *AFIPS Conference Proceedings* (1967), AFIPS Press.

[5] AVS 4.0 Developer's Guide and User's Guide, May 1992.

[6] Baker, M., Fox, G. C., and Yau, H. Cluster Computing Review. Tech. Rep. 1995, Center for Research on Parallel Computation, Nov. 1995.

[7] Baker, W. E., Bloom, S. C., Woollen, J. S., Nestler, M. S., and Brin, E. Experiments with a Three-Dimensional Statistical Objective Analysis Scheme Using FGGE Data. *Monthly Weather Review 115*, 1 (1987).

[8] Baumgardner, J. R., and Frederickson, P. O. Icosahedral Discretization of the Two-Sphere. *SIAM Journal of Numerical Analysis 22*, 6 (Dec. 1985), 1107–1115.

[9] Beguelin, A., Dongarra, J., Geist, G. A., Manchek, R., and Sunderam, V. A user's guide to PVM: Parallel virtual machine. Tech. Rep. TM-11826, Oak Ridge National Laboratory, 1991.

[10] Beguelin, A., Dongarra, J., Geist, G. A., Manchek, R., and Sunderam, V. A user's guide to PVM 3.0: Parallel virtual machine. Tech. Rep. TM-11826, Oak Ridge National Laboratory, 1993.

[11] Bic, L., and Gaudiot, J., Eds. *Special Issue: Dataflow Processing* (1990), vol. 10 of *Journal of Parallel and Distributed Computing.*

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

[13] Brouwer, A. E., Cohen, A. M., and Neumair, A. *Distance-Regular Graphs.* Springer Verlag, New York, 1989.

[14] Browne, J. C., Hyder, S. I., Dongarra, J., Moore, K., and Newton, P. Visual Programming and Debugging for Parallel Computing. *IEEE Parallel and Distributed Technology 3*, 1 (Spring 1995).

[15] Center for Analysis and Prediction of Storms. *Advanced Regional Prediction System*, version 3.0 ed. University of Oklahoma, Oct. 1992.

[16] Chandy, K. M., and Kesselman, C. Compositional C++: Compositional Parallel Programming. Tech. rep., California Institute of Technology, 1992.

[17] Chen, M., Cowie, J., Fox, G. C., Furmanski, W., and Rebbi, C. WebWork: Integrated Programming Environment Tools for National Grand Challenges. Tech. Rep. CRPC-TR95614, Center for Research on Parallel Computation, Rice University, June 1995.

[18] Cheng, D. Y. A Survey of Parallel Programming Languages and Tools. Tech. Rep. RND-93-005, NASA Ames Research Center, Moffet Field, CA, Mar. 1993.

[19] Cheng, G. *A Dataflow-based Software Integration Model in Parallel and Distributed Computing and Applications.* PhD thesis, Syracuse University, 1996.

[20] Choi, J., Dongarra, J. J., Pozo, R., and Walker, D. W. Scalapack: A scalable linear algebra library for distributed memory concurrent computers. In *Proceeding of the Fourth Symposium on the Frontiers of Massively Parallel Computation* (1992), IEEE Computer Society Press, pp. 120–127.

[21] The Connection Machine CM-5 Technical Summary, Oct. 1991.

[22] CODINE. http://www.genias.de/genias/english/codine/codine.html.

[23] Cohn, S. E., Sivakumarun, N., and Todling, R. Experiments with a Three-Dimensional Statistical Objective Analysis Scheme Using FGGE Data. *Monthly Weather Review* (Dec. 1994), 2838–2867.

[24] Cooperation, I. IBM Visualization Data Explorer (DX). http://www-i.almaden.ibm.com/dx/.

[25] Cray. *Cray Performance Optimization Manual.* man performance on a Cray.

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

[27] Data Online Monitoring System (DOLMS). http://dao.gsfc.nasa.gov/restricted_links/monitoring, Sept. 1996. Restricted access for GSFC.

[28] DaSilva, A. Personal communication, 1995.

[29] World Wide Web Cite of the Data Assimilation Office (DAO), NASA Goddard Space Flight Center. http://dao.gsfc.nasa.gov/restricted_links/monitoring, Sept. 1996. Restricted access for GSFC.

[30] Dennis, J. B. First Version of a Data Flow Procedure Language. vol. 19 of *Lecture Notes in Computer Science*, pp. 362–376.

[31] Dennis, J. B. A preliminary architecture for a basic data-flow Processor. In *ACM Proceedings of the Second Annual Symposium on Compute Architecture* (Jan. 1975), pp. 126–132.

[32] Dongara, J. Performance of various computers using standard linear equation software. Tech. Rep. CS-89-85, Oak Ridge National Laboratory, Oct. 1985,1996.

[33] El-Rewini, H., Lewis, T. G., and Ali, H. H. *Task Scheduling in Parallel and Distributed Systems.* Prentice Hall, 1994.

[34] ELIASSEN, A. Provisional Report on Calculation of Spatial Covariance and Autocorrelation of the Pressure Field. Tech. rep., Videnskaps-Akademiet, Institut for Vaer og Klimaforskning, Oslo, 1954.

[35] EXECUTIVE OFFICE OF THE PRESIDENT, OFFICE OF SCIENCE AND TECHNOLOGY POLICY. *A Research and Development Strategy for High Performance Computing*, Nov. 1987.

[36] FENTON, N. E. *Software Metrics: A Rigorous Approach*. Chapman and Hall, 1991.

[37] FENTON, N. E. Software Assessment: A necessary scientific basis. *Trns. Soft. Eng. 3*, 20 (1994), 199–206.

[38] FLANAGAN, D. *Java in a Nutshell*. O'Reiley, 1996.

[39] FLYNN, M. J. Some Computer Organizations and Their Effectiveness. *IEEE Trans. Computers C-21*, 9 (September 1972), 948–960.

[40] FORUM, H. P. F. High Performance Fortran Language Specification. Tech. rep., Rice University, 1993.

[41] FOSTER, I., AND CHANDY, K. M. Fortran M: A Language for Modular Parallel Programming. Tech. Rep. Preprint MCS-P237-0992, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, Ill., 1992, 1992.

[42] FOSTER, I., AND KESSELMAN, C. Globus: A Metacomputing Infrastructure Toolkit. Tech. rep., Argonne National Laboratory, 1996. http://www.globus.org.

[43] FOSTER, I., KESSELMAN, C., AND TUECKE, S. The Nexus Approach to Integrating Multithreading and Communication. Tech. rep., Argonne National Laboratory, 1996. http://www.globus.org.

[44] FOX, G. Parallel Computing in Industry: An Initial Survey. In *Proc. of Fifth Australian Supercomputing Conference* (World Congress Centre, Melbourne, Australia, Dec. 1992).

[45] Fox, G., and Furmanski, W. Towards Web / Java based High Performance Distributed Computing - an Evolving Virtual Machine. In *IEEE Conference HPDC-5* (Aug. 1996).

[46] Fox, G., Furmanski, W., Haupt, T., and Klasky, S. WebFlow: A Visual Problem Solving Environment for Wide-Area, Heterogeneous, Distributed High Performance Computing. NPAC Proposal to the NSF New Technologies Program, July 1996.

[47] Fox, G., Johnson, M., Lyzenga, G., Otto, S., Salmon, J., and Walker, D. *Solving Problems on Concurrent Processors.* Prentice Hall, New Jersey, 1988.

[48] Fox, G. C. Parallel Computers and Complex Systems. In *Complex Systems '92: From Biology to Computation* (1992), Bossomaier and D. G. Green, Eds., Inaugural Australian National Conference on Complex Systems. also CRPC-TR92266.

[49] Fox, G. C., and et al. InfoMall: A Scalable Organization for the Development of HPCC Software and Systems. Tech. Rep. SCCS-531, NPAC at Syracuse University, Oct. 1993.

[50] Fox, G. C., and et al. InfoVision: Information, Video, Imagery and Simulation on Demand. Tech. Rep. SCCS-575, NPAC at Syracuse University, 1993.

[51] Fox, G. C., and Furmanski, W. Factoring on the World Wide Web Computing Project. http://www.npac.syr.edu/factoring.html, 1995.

[52] Fox, G. C., and Furmanski, W. SNAP, Crackle, WebWindows! Tech. Rep. SCCS758, NPAC at Syracuse University, 1996.

[53] Fox, G. C., Hiranadani, S., Kennedy, K., Koelbel, C., Kremer, U., Tseng, C.-W., , and Wu, M.-Y. Fortran D Language Specification. Tech. Rep. SCCS-42c, NPAC at Syracuse University, 1991. Rice University,TR90-141.

[54] Fox, G. C., Williams, R. D., and Messina, P. C. *Parallel Computing Works.* Morgan Kaufmann, 1994. http://www.npac.syr.edu/copywrite/pcw.

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

[56] GOLUB, G. H., AND LOAN, C. F. V. *Matrix Computations*. John Hopkins University Press, 1989.

[57] GUO, J., AND DA SILVA, A. 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.

[58] HAUPT, T., HAWICK, K., AND MACIVIC, M. Evaluation of HPF Compilers. Personal communication, 1992/93.

[59] HAUPT, T., AND KLASKY, S. Implementation of the T2 code in HPF. http://www.npac.syr.edu/users/haupt/bbh/HPF/index.html, 1996.

[60] HEIKES, R. P. The Shallow Water Equations on a Spherical Geodesic Grid. Tech. Rep. 524, Department of Atmospheric Science Colorado State University, 1993.

[61] PVM and HeNCE Programmers Manual. http://www.epcc.ed.ac.uk/epcc/-publications/cray/pvm.hence.30/index.html.

[62] HENDERSON-SELLERS, A., HENDERSON-SELLERS, B., POLLARD, D., VERNER, J., AND PITMAN, A. Applying Software Engineering Metrics to Land Surface Parameterization Schemes. *Journal of Climate 8* (May 1995).

[63] HOARE, C. A. R. *Communicating Sequential Processes*. Prentice Hall, 1985.

[64] HUDSON, A. Reference to Performance Data Automatically Collected on the GSFC Cray. NASA GSFC, Data Assimilation Office, Personal Communication, Jan. 1996.

[65] HWANG, K., AND BRIGGS, F. A. *Computer Architecture and Parallel Processing*. McGraw-Hill, 1986.

[66] 4th International Symposium on Solving Irregular Structured Problems in Parallel, June 1997.

[67] ISAKSEN, L. Parallelizing the ECMWF Optimum Interpolation analysis. In *Parallel Supercomputing in Atmospheric Science, Proceedings of the Fifth ECMWF Workshop on the Use of Parallel Processors in Meteorology* (Nov. 1992).

[68] Jagannathan, R. *Parallel and Distributed Computing Handbook.* McGrawhill, 95, ch. Dataflow Models.

[69] Java-Linux. http://substance.blackdown.org/java-linux.html, Oct. 1996.

[70] Kahn, G. A Semantic of a Simple Language for Parallel Processing. In *Proceedings IFIP Congress* (Amsterdam, 1974), Elsvier North Holland, pp. 471–475.

[71] Karp, R. M., and Miller, R. E. Properties of a Model for Parallel Computations: Determinacy, Termination, Queuing. *SIAM Journal of Applied Mathematics 6*, 14 (1966), 1390–1411.

[72] Khanna, R., Ed. *Distributed Computing.* Prentice Hall, 1994.

[73] http://www.khoros.unm.edu/khoros/khoros2/home.html.

[74] Kowalczyk, E. A. A soilcanopy scheme for use in a numerical model for the atmosphere – 1D standalone model. Tech. Rep. 23, CSRIO, DAR, 1991.

[75] Kurihara, Y. A Finite Difference Scheme by Making Use of Primitive Equations of a Spherical Grid. *Monthly Weather Review 93* (69), 399–415.

[76] Legion. http://www.cs.virginia.edu/ legion, 1996.

[77] Lemay, L., and Perkins, C. L. *Teach Yourself Java in 21 Days.* Sams Net, 1996.

[78] Lippman, S. B. *C++ Primer*, 2nd ed. Addison-Wesley, 93.

[79] Lumsdaine, A., Squere, J., and McCandless, B. *Object Oriented MPI (OOMPI): A C++ Class Library for MPI.* University of Notre Dame, July 1996.

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

[81] MAKIVIĆ, M., AND VON LASZEWSKI, G. High-Performance Computing and Four-Dimensional Data Assimilation: The Impact on Future and Current Problems. Tech. Rep. Final Report, NPAC at Syracuse University, Aug. 1996.

[82] *MPI – Document for a Standard Massage-Passing Interface.* University of Tennesse, Nov. 1993.

[83] MPI: A Message-Passing Interface Standard. http://www.mcs.anl.gov/mpi/mpi-report/mpi-report.html, May 1994.

[84] NASA DATA ASSIMILATION OFFICE AT GODDARD SPACE FLIGHT CENTER. *Data Assimilation Program Version 1.2.* Greenbelt, MD, 1993. The version 1.2s has been made available by Mike Seablom.

[85] NASA DATA ASSIMILATION OFFICE AT GODDARD SPACE FLIGHT CENTER. *Data Assimilation Program Version 2.0.* Greenbelt, MD, 1993. The version 2.0mv has been made available by David Lamich.

[86] NEWTON, P. Visual Programming and Parallel Computing. In *Workshop on Environments and Tools for Parallel Scientific Computing* (Walland, TN, May 1994).

[87] OUSTERHOUT, J. *Tcl and the Tk Toolkit.* Adisson Wessley, 1994.

[88] PFAENDTNER, J., BLOOM, S., LAMICH, D., SEABLOM, M., SIENKIEWICZ, M., STOBIE, J., AND DA SILVA, A. 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).

[89] PFAENDTNER, J., ROOD, R., SCHUBERT, S., BLOOM, S., LAMICH, D., SEABLOM, M., AND SIENKIEWICZ, M. The Goddard Global Data Assimilation System: Description and Evaluation. *Submitted to Mon. Wea. Review* (1993).

[90] PFAENTDNER, J. The use of Icosahedral Grids in PSAS. Working note, Goddard Space Flight Center, Seabrook, MD., 1992.

[91] RICHARDSON, L. *Weather Prediction by Numerical Process.* Cambridge University Press, 1922.

[92] Roskies, R. Metacomputing - Pipedream or Practical Reality. *Computers in Physics 8*, 5 (Sept/Oct 1994), 540–545.

[93] Sabot, G., and Wholey, S. Parallel execution of a Fortran 77 Weather Prediction Model. In *Supercomputing 93* (Nov. 1993).

[94] Saltz, J., Das, R., Ponnusamy, R., Mavriplis, D., Berryman, H., and Wu, J. PARTI procedures for realistic loops. In *Proceedings of the 6th Distributed Memory Computing Conference* (Portland, OR, April-may 1991).

[95] Seablom, M. 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* (Clermont-Ferrand, France, 1990), vol. WMO. Preprint volume, pp. 628–630.

[96] Sedgewick, R. *Algorithms*. Addison Wesley, 1992.

[97] Silicon Graphics Inc. *Iris Explore User's Guide*, 1992.

[98] Silva, A. D., Pfaendtner, J., Guo, J., Sienkiewicz, M., and Cohn, S. E. 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).

[99] Smarr, L. L., and Catlett, C. E. Metacomputing. *Communication of the ACM 35*, 6 (June 1992), 45–52.

[100] Snir, M., Otto, S. W., Huss-Lederman, S., Walker, D. W., and Dongara, J. *MPI: The Complete Reference*. Scientific and Engineering Computation Series. The MIT Press, 1996.

[101] Stobie, J. Personal communication, Nov. 1995.

[102] Stobie, J. G. Correlated Instrument Errors in Optimal Interpolation (OI) Data Assimilation. *Monthly Weather Review* (Sept. 1994). (submitted).

[103] Stone, H. S. *High-Performance Computer Architecture*. Addison Wessley, 1987,1995.

[104] TAKACS, L. L., MOLOD, A., AND WANG, T. 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).

[105] TANNENBAUM, A. S. *Distributed Operating Systems.* Prentice Hall, 1995.

[106] THOMSON, J. F., WARSI, Z. U. A., AND MASTIN, C. W. *Numerical Grid Generation.* North-Holland, 1985.

[107] TRENBRETH, K. E., Ed. *Climate System Modeling.* Cambridge University Press, 1993.

[108] TREW, A., AND EDS., G. W. *Past, Present, Future.* Springer, 1991.

[109] UNIVERSITY OF VIRGINIA. *The Mentat Project.* http://www.cs.virginia/ mentat.

[110] UPSON, C., FAULHABER, T., JR., KAMINS, D., LAIDLAW, D., SCHLEGEL, D., VROOM, J., GURWITZ, R., AND VAN DAM, A. The Application Visualization System: A Computational Environment for Scientific Visualization. *IEEE Computer Graphics and Applications* (July 1989).

[111] VAN ROSSUM, G. *Python Tutorial.* Dept. CST, CWI, Amsterdam, NL, 1994.

[112] VON LASZEWSKI, G. A parallel genetic algorithm for the graph partitioning problem. In *Transputer Research and Applications 4, Proc. of the 4th Conf. of the North-American Transputers Users Group* (Ithaca, NY, 1990), IOS Press.

[113] VON LASZEWSKI, G. Intelligent Structural Operators for the k-way Graph Partitioning Problem. In *Proc. of the 4th intern. Conf. on Genetic Algorithms* (San Diego, CA, July 1991), Morgan Kaufman. plenary presentation.

[114] VON LASZEWSKI, G. A Collection of Graph Partitioning Algorithms: Simulated Annealing, Simulated Tempering, Kernighan Lin, Two Optimal, Graph Reduction, Bisection. Tech. Rep. SCCS 477, Northeast Parallel Architectures Center at Syracuse University, Apr. 1993.

[115] VON LASZEWSKI, G. Implementing the Advanced Regional Prediction System (ARPS) with Fortran D. Tech. Rep. SCCS 492, Northeast Parallel Architectures Center at Syracuse University, June 1993.

[116] VON LASZEWSKI, G. Issues in Parallel Computing. Tech. Rep. SCCS 577, Northeast Parallel Architectures Center at Syracuse University, Dec. 1993.

[117] VON LASZEWSKI, G. Object Oriented Recursive Bisection on the CM-5. Tech. Rep. SCCS 476, Northeast Parallel Architectures Center at Syracuse University, Apr. 1993.

[118] VON LASZEWSKI, G. Parallelization of MOPAC. Tech. Rep. SCCS 577, Northeast Parallel Architectures Center at Syracuse University, May 1993.

[119] VON LASZEWSKI, G. Preliminary Performance of a Parallel Interpolation Algorithm. Tech. Rep. SCCS 713, Northeast Parallel Architectures Center at Syracuse University, Dec. 1995. first edition in June 1995.

[120] VON LASZEWSKI, G. Interactive Parallel Program Generation. In *Making its Mark: Proceedings of the 6th Workshop of The use of Parallel Processors in Meteorology, European Centre for Medium Weather Forecast, Reading, UK* (Dec. 1996), G.-R. Hoffman and N. Kreitz, Eds., World Scientific. to be published.

[121] VON LASZEWSKI, G., AND ET AL. Design Issues for the Parallelization of an Optimal Interpolation Algorithm. In *Coming of Age: Proceedings of the 4th Workshop on the Use of Parallel Processing in Atmospheric Science, European Centre for Medium Weather Forecast, Reading, UK* (Nov. 1994), G.-R. Hoffman and N. Kreitz, Eds., World Scientific, pp. 290–302. (ftp)

[122] VON LASZEWSKI, G., AND MACIVIC, M. The Four Dimensional Data Assimilation Web Page. http://www.npac.syr.edu/projects/nasa.

[123] VON LASZEWSKI, G., MACIVIĆ, M., LYSTER, P., DASILVA, A., LAMICH, D., AND DEE, D. Problems with the Quality Control. Meeting at NASA GSFC, June 1996.

[124] VON LASZEWSKI, G., MOHAMED, A. G., AND FOX, G. C. Blocked LU Factorization on a Multiprocessor Computer. *Microcomputer in Civil Engineering 8*, 1 (1993), pp. 45–56.

[125] VON LASZEWSKI, G., AND MÜHLENBEIN, H. A Parallel Genetic Algorithm for the k-way Graph Partitioning Problem. In *1st inter. Workshop on Parallel Problem Solving from Nature* (University Dortmund, West Germany, Nov. 1990), Springer, Ed.

[126] VON LASZEWSKI, G., PARASHAR, M., MOHAMED, A. G., AND FOX, G. C. High Performance Scalable Matrix Algebra Algorithms for Distributed Memory Architectures. In *Proceedings of Supercomputing 92* (Minneapolis, Nov. 1992), IEEE Compt. Soc. Press, pp. 170–179. Overall Best Student Paper Award.

[127] WALL, L., AND SCHWARTZ, R. L. *Programming Perl.* O'Reiley, 1992.

[128] WAMM, Wide Area Metacomputer Manager. http://miles.cnuce.cnr.it/pp/wamm, 1996.

[129] WILLIAMSON, D. L. Review of Numerical Approaches for Modeling Global Transport. In *Air Pollution Modeling and its Application*, H. van Dop and G. Kallos, Eds. Plenum Press, 1992.

[130] XPVM. http://www.lncc.br/tutorials/SP1.Training/Maui.training/workshop/html/-xpvm/XpvmExercise.html.

Vita

Name:	Gregor von Laszewski
Date of Birth:	June 8, 1965
Place of Birth:	Bonn, Germany

Elementary School: Grundschule Niederpleis, Sankt Augustin, Germany

High School: Albert-Einstein Gymnasium, Sankt Augustin, Germany
– Graduated 1984

Universities:

- University of Bonn, Germany
  – BS, in computer and information science with minors mathematics and physics (1987)
  – MS, in computer and information science with minor physics (1989)
- University Fellow at The Ohio State University (1990)
- Syracuse University, Syracuse, New York (1991)

Affiliations:

- German National Research Centre for Infomations Technology (GMD)
- Northeast Parallel Architectures Center at Syracuse University
- Data Assimilation Office at NASA Goddard Space Flight Center

Awards:

- Fellowship award from the University of Bonn (1989)
- Financial support by the German Government due to outstanding grades (1990)
- Fellowship award at The Ohio State University (1990)
- Overall best student paper at Supercomputing (1992)
- Member of the program committee of Supercomputing (1993)
- Sponsored by University Space Research Agency (1995)