DAO Office Note 1998-008

Office Note Series on Global Modeling and Data Assimilation

Richard B. Rood, Head
Data Assimilation Office
Goddard Space Flight Center
Greenbelt, Maryland

PILGRIM: A Parallel Library for Grid Manipulations in Earth Science Calculations

William Sawyer*, Lawrence Takacs*, Arlindo da Silva*, Peter Lyster *

Data Assimilation Office, Goddard Laboratory for Atmospheres
* *Goddard Space Flight Center, Greenbelt, Maryland*

Goddard Space Flight Center
Greenbelt, Maryland 20771
May 1998

Abstract

This office note discusses the design of PILGRIM, a library which will support the manipulations of grids in Earth Science software currently being designed and implemented at the Data Assimilation Office (DAO). It allows various grids to be distributed over an array of processing elements (PEs) and manipulated with high parallel efficiency.

Unlike many parallel libraries, the developer takes some basic responsibility for laying out the data distribution. This adds to the simplicity of the implementation. Moreover, developers generally *want* some control of and understanding about the data's distribution. This is particularly true in the DAO's applications, e.g., GEOS DAS, in which data distributions are known in advance.

PILGRIM has three distinct layers: low-level facilities to perform communication as well as basic linear algebra operations on distributed vectors, transformation kernels which work on distributed sparse matrices, and modules which define grid types and operations on those grids.

The design of PILGRIM closely follows the requirements of GEOS DAS. On the other hand, the library is being designed in a way that ensures it can support other applications which employ certain types of grids. It can also be extended with new grid modules to support still others.

Contents

List of Figures

1 Introduction

The need to discretize continuous models in order to solve scientific problems gives rise to finite *grids* — a set of points at which prognostic variables are sought. So prevalent is the use of grids in science that we often forget that a computer-calculated solution is not the solution to the original problem but rather of a discretized representation of the original problem, and is only an approximative solution at that, due to finite precision arithmetic. Grids are ubiquitous where analytical solutions to continuous problems are not obtainable, e.g., the solution of many differential equations.

Classically a structured grid is chosen a priori for a given problem. If the quality of the solution is not acceptable, then the grid is made finer, in order to better approximate the continuous problem. More recently the practicality of *unstructured* grids has been recognized. In such grids it is possible to cluster points in regions of the domain which require higher resolution, while retaining coarse resolution in other parts of the domain.

Unstructured grids are often employed in device simulation [1], computational fluid dynamics [2], and even in oceanographic models [3]. Although these grids are more difficult to lay out than structured grids, much research has been done in generating them automatically [4], and thus this disadvantage is being alleviated. In addition, once the grid has been generated, there are numerous methods and libraries to adaptively refine the mesh [5] to provide a more precise solution.

Furthermore, the advantages of multiple grids of varying resolutions for a given domain have been recognized. This is best known in the Multigrid technique [6] in which low frequency error components of the discrete solution are eliminated if values on a given grid are restricted to a courser grid on which a smoother is applied. But multiple grids also find application other fields such as speeding up graph partitioning algorithms [7].

An additional complication has arisen in the last few years: many contemporary scientific problems must be decomposed over an array of processing elements (or PEs) in order to calculate a solution in an expedient manner. Depending on the parallelization technique, not only the work load but also the grid itself may be distributed over the PEs, meaning that different parts of the data reside in completely different memory areas of the parallel machine. This makes the programming of such an application much more difficult for the developer.

The Goddard Earth Observing System (GEOS) Data Assimilation System (DAS) software currently being developed at the Data Assimilation Office (DAO) is no exception to the list of modern grid applications. GEOS DAS uses observational data of uncertain accuracy and incomplete global coverage to estimate the complete, evolving dynamic, energetic and constituent state of the global earth system. The GEOS DAS consists of two main components, an atmospheric General Circulation Model (GCM) [8] to predict the time evolution of the global earth system and a Physical-space Statistical Analysis Scheme (PSAS) [9] to periodically incorporate observational data.

At least three distinct grids are being employed in GEOS DAS: an *observation grid* — an unstructured grid of points at which observed or measured physical quantities from instruments or satellites are associated — a structured *geophysical grid* of points spanning the earth at uniform latitude and longitude locations at which prognostic quantities are determined and a block-structured *computational grid* which may be stretched in latitude and longitude on which the dynamical calculation takes place. Each of these grids has a different structure and number of constituent points, but there are numerous interactions between the grids. Finally the GEOS DAS application is targeted for distributed memory architectures and employs a message-passing paradigm for the communication between PEs.

In this document we describe the design of PILGRIM, a library for grid manipulations, which will

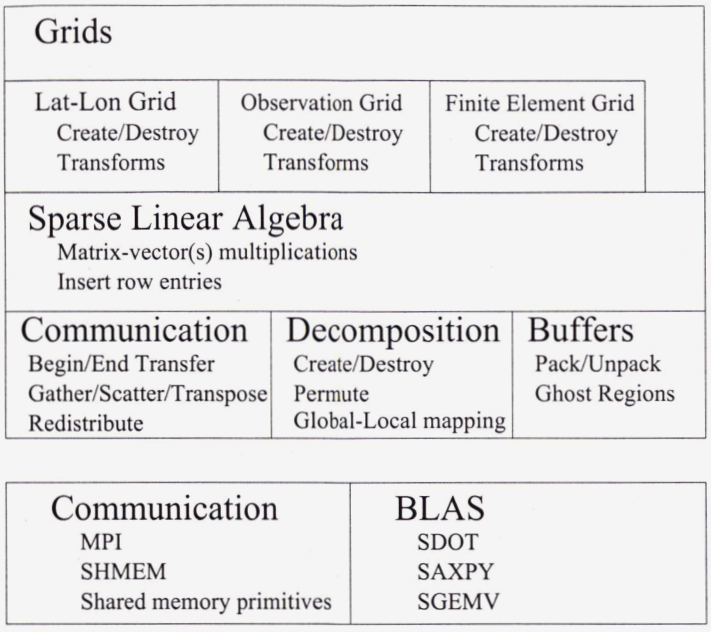

Figure 1: PILGRIM assumes the existence of fundamental communication primitives and optimized BLAS. At its lowest level, PILGRIM supplies communication subroutines and sparse Linear Algebra utilities. Above that transformation kernels are built. Finally, at the highest level there are grid module which describe the nature of the grid, its decomposition and operations on that grid.

meet the requirements of GEOS DAS. The design is layered as indicated in Figure 1. Communication primitives such as the Message-Passing Interface (MPI) and Basic Linear Algebra Subroutines or BLACS [10] are assumed. The first layer contains a module for high-level communication routines as well as modules to define the decomposition of the domain, and to pack and unpack sub-regions of the local domain. Above this layer is a sparse linear algebra layer which performs basic sparse matrix operations for grid transformations. The final layer contains plug-in modules, each supporting a different grid. This layer is extensible — these modules make use of the lower layers and new grids, together with the operations performed on them, can be implemented as the need arises.

The design of PILGRIM is *object-oriented* [11] in the sense that it is modular, data is encapsulated in each design layer, operations can be overloaded, and different instantiations of grids can coexist simultaneously. The library is realized in Fortran 90 which partially supports an object-oriented paradigm. It also allows the necessary software engineering techniques, e.g., modules and derived data types, while keeping the in line with other Fortran developments at the DAO. The communication layer is implemented using MPI [12], however the communication interfaces defined in PILGRIM's primary layer could conceivably be implemented with other message-passing libraries, e.g., PVM [13], or with other paradigms, e.g., Cray SHMEM [14] or even with shared-memory primitives which are available on shared-memory machines like the SGI Origin or SUN Enterprise.

This document is structured in a bottom-up fashion: in Section 2 requirements for PILGRIM are derived from GEOS DAS requirements. Reasonable design assumptions are made in Section 3 in order to ease the implementation. The layer for communication, decompositions, and buffer packaging is discussed in Section 4. The transformation kernel layer is specified in Section 5. The plug-in grid modules are defined in Section 6 to the degree foreseen for the implementation of GEOS DAS. In Section 7 some examples and prototype benchmarks are presented for the interaction of all

the components. Finally we summarize our work on PILGRIM in Section 8.

2 Requirements

The following requirements for PILGRIM are derived requirements from the GEOS DAS component requirement documents [15, 16].

1. Data arrays (possibly multidimensional) need to be distributed over an array of PEs, as determined by an application at run-time.

   The data distribution is *not* transparent, and the application will decide what decomposition is best. This can be an advantage as the application may use this knowledge in order to make run-time decisions, e.g., load balancing.

2. The library has to support different distributions of the same grid, and several different types of grids to represent observations, the geophysical view of the earth, and the mesh used to perform the dynamical calculation.

3. The library should be extensible, i.e., new grids can be defined as needed. The addition of a new grid to the library should only require linking in a new module to the existing library.

4. It is necessary to support the redistribution of grids at run-time for load balancing and other reasons.

5. The application must be able to define linear transformations from one grid to another. Again, grids do not need to have the same number of grid-points or a similar distribution.

6. Even though the application defines the data decompositions for the grids and two such decompositions might not be mutually ideal for a given transformation. The library should make an attempt to set up the transformation in such a way as to maximize performance when it is applied.

7. The use of the PILGRIM library cannot change the underlying algorithms used in GEOS DAS. That is, the numerical results when the library is used should have at most round-off differences to the current sequential GEOS DAS results.

A literature search was the first step taken in the PILGRIM design process to find public domain libraries which might be sufficient for the DAO's needs. Surprisingly, none of the common parallel libraries for the solution of sparse matrix problems, e.g., PETSc [17], Aztec [18], PLUMP [19] et al., was sufficient for our purposes. These libraries all try to achieve transparency the parallel implementation to the application developer. In particular, the application is not supposed to know how the data are actually distributed over the PEs.

This trend in libraries is not universally applicable for the simple reason that if an application is to be parallelized, the developers generally have a good idea of how the underlying data should be distributed and manipulated. Experience has shown us that hiding complexity often leads to poor performance, and the developer often resorts to workarounds to make the system to perform in the manner she or he envisions. If the developer of a parallel program is capable of deciding on the proper data distribution and manipulation of local data, then those decisions need to be supported.

3 Basic Design Assumptions

In order to minimize the scope of PILGRIM, we make some simplifying assumptions about the way the library will be used. These assumptions do not conflict with the GEOS DAS requirements.

1. The local portion of the distributed grid array is assumed to be a contiguous section of memory. The local array can have any rank, but if the rank is greater than one, the developer must assure that no gaps are introduced into the actual data representation (i.e., by packing it into a 1-D array if necessary).
2. Grid transformations are assumed to be *sparse* linear combinations of few grid point values of one grid to a given point value on the other. The linear transformation corresponds to a sparse matrix with a predictable number of non-zeros per row. This assumption is realistic for the localized interpolations used in GEOS DAS.
3. At a high level, the application can access data through global indices (i.e., indices of the original, undistributed problem). However, at the level where most computation is performed the application needs to work with local indices (ranging from one to the total number of elements in the local contiguous array). The information to perform global-to-local and local-to-global mappings must be contained in the data structure defining the grid. However, it is assumed that these mappings are performed seldomly, e.g., at the beginning and end of execution, and these mappings need not be efficient.
4. All decomposition-related information is replicated on all PEs.

These assumptions are significant. The first avoids the introduction of an opaque type for data and allows the application to manipulate the local data as it sees fit. The fact that the data is contained in a simple data structure generally allows higher performance than an implementation which buries the data inside a derived type. The second assumption ensures that the grid transformations are not memory limited. The third implies that most of the calculation is performed addressing the data in a *local* fashion. In GEOS DAS it is fairly straightforward to run in this mode, however it might not be the case in other applications. The last assumption assures that every PE knows about the entire data decomposition. Since the data are independent of the decomposition, memory requirements for this replication are not an issue.

4 Communication and Decomposition Utilities

In this layer we isolate functionality which is machine or message-passing-paradigm dependent, or which is required by a large number of modules. Among the non-communication-related utilities operations for moving sections of data arrays to and from buffers are provided, as well as utilities to define and use data decompositions.

4.1 Data Decomposition Utilities

The operations on data decompositions are embedded in a Fortran 90 module, which also supplies generic `DecompType` to describe a decomposition. Any instance of `DecompType` is replicated on all PEs such that every PE has access to information about the entire decomposition. The decomposition utilities consist of the following:

DecompRegular1d	Create a 1-D blockwise data decomposition
DecompRegular2d	Create a 2-D block-block data decomposition
DecompRegular3d	Create a 3-D block-block-block data decomposition
DecompIrregular	Create an irregular data decomposition
DecompCopy	Create a new decomposition with contents of another
DecompPermute	Permute the assignment of PEs in a given decomposition
DecompFree	Free a decomposition and the related memory
DecompGlobalToLocal1d	Map global 1-d index to local (pe,index)
DecompGlobalToLocal2d	Map global 2-d index to local (pe,index)
DecompLocalToGlobal1d	Map local (pe,index) to global 1-d index
DecompLocalToGlobal1d	Map local (pe,index) to global 2-d index

The calls which create a new decomposition can be overloaded to `DecompCreate` and, likewise, the 1-D and 2-D global-to-local and local-to-global mappings are denoted by `DecompGlobalToLocal` and `DecompLocalToGlobal`, resulting in five fundamental operations.

4.2 Communication Utilities

Communication primitives are defined in this layer for the following reasons: it may be necessary at some point to implement communication primitives directly with a message-passing library other than MPI, e.g., PVM or SHMEM, or even with shared-memory primitives such as those on the SGI Origin, which is the principle platform at the DAO. Thus it is wise to encapsulate all message-passing into one module. The communication utilities constitute a Fortran 90 module. For brevity, only the overloaded functionality is presented:

ParInit	Initialize the parallel code segment
ParExit	Exit from the parallel code segment
ParSplit	Split parallel code segment into two groups
ParMerge	Merge two code segments
ParScatter	Scatter global array to given data decomposition
ParGather	Gather from data decomposition to global array
ParGhost	Ghost a boundary regions of a regular grid
ParBeginTransfer	Begin asynchronous data transfer
ParEndTransfer	End asynchronous data transfer
ParExchangeVector	Transpose block distributed vector over all PEs
ParRedistribute	Redistribute one data decomposition to another

4.3 Buffer Utilities

In order to perform calculations local to a given PE it is often necessary to "ghost" adjacent regions, that is, send boundary regions of the local domain to adjacent PEs. For the grids used in GEOS DAS, we have defined a module to move ghost regions to and from buffers. The buffers can be transferred to another PEs with the communication primitives, e.g., `ParBeginTransfer` and `ParEndTransfer`. Currently, the functionality in this buffer module is as follows:

BufferPackGhost2dReal	Pack a 2-D array sub-region into buffer
BufferUnpackGhost2dReal	Unpack buffer into 2-D array sub-region
BufferPackGhost3dReal	Pack a 3-D array sub-region into buffer
BufferUnpackGhost3dReal	Unpack buffer into 3-D array sub-region
BufferPackSparseReal	Pack specified entries of vector into buffer
BufferUnpackSparseReal	Unpack buffer into specified entries of vector

These operations are overloaded to `BufferPack` and `BufferUnpack`.

In this module, as with most others, the indices local to the PE are used. Clearly this puts responsibility on the programmer to keep track of the indices which correspond to the ghost regions. In GEOS DAS this is fortunately fairly straightforward.

5 Sparse Linear Algebra

The concept of transforming one grid to another means to perform an interpolation of the values defined on one set of grid-points to values defined on another set. Irrespective of the rank of the grid, these values are stored as vectors with a given length and distribution, as defined by the grid decomposition. Thus this layer is little more than facility to perform linear transformations on distributed vectors.

A vector is a contiguous array of REALs which are addressed from $1 \ldots N_{local}$ i.e., local indices ranging from the first to the last local entry. The contiguous array is one-dimensional although it might actually represent a multi-dimensional array at a higher level. Local indices are used when referring to individual matrix entries, but the mappings `DecompGlobalToLocal` can be used for the translation from global to local indices.

As in all parallel sparse linear algebra packages, e.g., Aztec and PETSc, the linear transformation is stored in a distributed sparse matrix format. The application of the linear transformation is a matrix-vector multiplication where the matrix is not necessarily square, and the resulting vector may be distributed in a different manner than the original.

There are many approaches to storing distributed sparse matrices and many ways to perform a the matrix-vector product. PILGRIM uses a format similar to that described in [19], which is optimal if the number of non-zeros per row is constant.

5.1 Matrix Creation and Definition

Assumption 3 in Section 3 implies that the matrix definition is not time-consuming. For example in GEOS DAS, the template of any given interpolation is initialized once, but the interpolation itself is performed repeatedly. Thus fairly little attention has been paid to the optimization of the matrix creation and definition. The basic operations for creating and storing matrix entries are:

SparseMatCreate	Create a sparse matrix
SparseMatDestroy	Destroy a sparse matrix
SparseInsertEntries	Insert entries replicated on all PEs
SparseInsertLocalEntries	Insert entries of local PE

Two scenarios for inserting entries are supported. In the first scenario, every processor inserts all matrix entries. Thus every argument of the corresponding routine, `SparseInsertEntries`, is replicated. The local PE picks up only the data which it needs, leaving all other data to the appropriate PEs. This scenario is the easiest to program if the sequential code version is used as the code base.

In the second scenario the domain is partitioned over the PEs, that is, each PE is responsible for a disjoint subset of the matrix entries, and the matrix generation is performed in parallel. Clearly this is the more efficient scenario. The corresponding routine, `SparseInsertLocalEntries` assumes only that no two PEs try to add the same matrix entry. However, it does not assume that the all matrix entries reside on the local PE, and it will perform the necessary communication to put the matrix entries in their correct locations.

5.2 Matrix Multiplication

The efficient application of the matrix to a vector or group of vectors is crucial to the overall performance of GEOS DAS, since the linear transformations are performed continually on assimilation runs executing for days or weeks at a time.

The most common transformation is between three-dimensional arrays of two different grids which describe global atmospheric quantities such as wind velocity or temperature. One 3-D array might be correspond to the geophysical grid which covers the globe, while another might be the computional grid which is more appropriate for the dynamical calculation. The explicit description of such a 3-D transformation would be prohibitive in terms of memory. But fortunately, all the transformations performed in GEOS DAS have dependencies in only two of the three dimensions, for example, the transformation of 2-D horizontal cross-sections of the geophysical and computational grids. Matrices therefore correspond to these transformations of 2-D arrays, however, the transformation might be applied to any number of 2-D arrays, for example all vertical levels of a physical quantity.

To fulfill the requirements of GEOS DAS, a 2-D array is unwrapped to a vector x to which the matrix is applied. Using this representation the transformation become simple matrix-vector multiplications. The following two operations which realize matrix-vector and a matrix-transpose-vector multiplications:

SparseMatVecMult	Perform $y \leftarrow \alpha Ax + \beta y$
SparseMatTransVecMult	Perform $y \leftarrow \alpha A^T x + \beta y$

In order to transform several arrays simultaneously, the arrays are grouped into multiple vectors, i.e., into a $n \times m$ matrix where n is the length of the vector (number of values in the array), and m is the number of vectors. The following matrix-matrix and matrix-transpose-matrix multiplications can group messages in such a way as to drastically minimize latencies and utilize more efficient BLAS-2 operations:

SparseMatMatMult	Perform $Y \leftarrow \alpha AX + \beta Y$
SparseMatTransMatMult	Perform $Y \leftarrow \alpha A^T X + \beta Y$

The distributed representation of the matrix contains, in addition to the matrix information itself, space for the *communication pattern*. Upon entering any one of the four matrix operations, a check is made on the consistency of the matrix, i.e., whether new entries to the matrix have been added since

its last application. If the matrix has been modified, the operation first generates the communication pattern — an optimal map of the information which has to be exchanged between PEs — before performing the matrix operation. This is a fairly expensive operation, but in GEOS DAS it only needs to be done once when the matrix is first defined. After that, the matrix operation can be repetitively performed in the most efficient manner possible.

6 Supported Grids

The *grid data structure* describes a set of *grid-points*, a decomposition of the grid-points over a group of PEs, and other information, for example, the size of the domain. The grid data structure itself does not contain data, and, since it does not require much memory, is replicated on all PEs. Data are contained in other arrays, which are *distributed* over the PEs, and are given meaning by the information in the grid data structure.

For example, the temperature on a regular three-dimensional geophysical grid in GEOS DAS might be held in a 3-D array. The decomposition of the this array might be into vertical columns, i.e., a block-block decomposition of the two-dimensional cross-section of the array. The grid data structure contains the information to describe the grid and its decomposition, and is replicated on all PEs. The data itself is a 3-D array with the dimensions of the column local to that PE (information which the application must supply for the definition of the decomposition). That is, the 3-D temperature array is *local* to the PE and can be manipulated by the application as it sees fit.

6.1 Latitude-Longitude Grid

This grid corresponds to a *lat-lon* coordinate system, i.e., a regular grid with all points in one row having a given latitude and all points in a column a given longitude. The grid encompasses the entire earth from $-\pi$ to π longitudinally and from $-\pi/2$ to $\pi/2$ in latitude. The $\Delta\lambda$ and $\Delta\Phi$ are constant for a given latitude and longitude respectively.

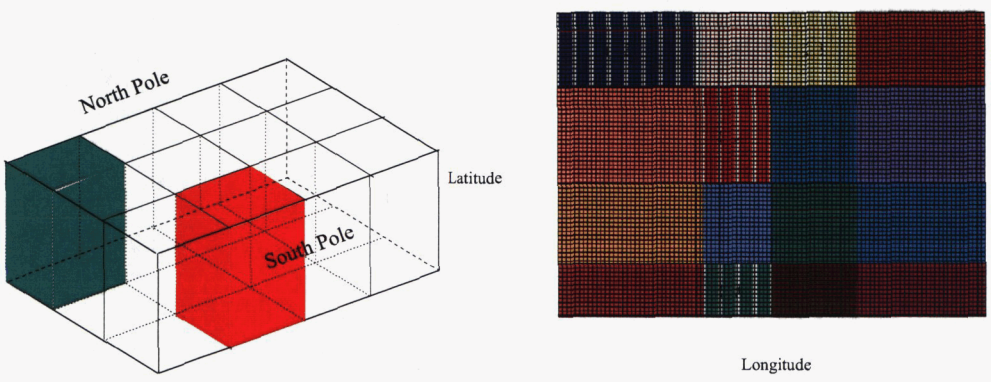

Figure 2:

The decomposition of this grid is in a checkerboard fashion, with the participating PEs mapped into a Cartesian coordinate system (see Figure 2). That is, the grid decomposition is two dimensional: the underlying three-dimensional data comprises all levels of a column of data designated by the 2-D decomposition of the horizontal cross-section. One PE gets a sub-block of the entire grid. This

decomposition can contain a variable sized rectangle of points — it is not necessary for each PE to be assigned an equal number of points, and thus some freedom for load balancing is available.

The basic data structure for the lat-lon grid is the following:

```
TYPE LatLonGridType
  TYPE (DecompType)    :: Decomp          ! Decomposition
  INTEGER              :: ImGlobal        ! Global Size in X
  INTEGER              :: JmGlobal        ! Global Size in Y
  REAL                 :: Tilt            ! Tilt of remapped NP
  REAL                 :: Rotation        ! Rotation of remapped NP
  REAL                 :: Precession      ! Precession of remapped NP
  REAL,POINTER         :: dLat(:)         ! Latitudes
  REAL,POINTER         :: dLon(:)         ! Longitudes
END TYPE LatLonGridType
```

This grid can adequately describe both the GEOS DAS *computational* grid used for dynamical calculations, and the geophysical grid in which the prognostic variables are sought. The former makes use of the parameters **Tilt**, **Rotation** and **Precession** to describe its view of the earth (see Figure 3), and the **dLat** and **dLon** grid box sizes to describe the grid stretching. The latter is denoted by the normal geophysical values for **Tilt**, **Rotation** and **Precession** $= (\frac{\pi}{2}, 0, 0)$ and uniform **dLat** and **dLon**.

6.2 Observation Grid

The observation grid data structure describes observation points over the globe, as described by their lat-lon coordinates. In contrast to the lat-lon grid, the point grid decomposition is inherently *one-dimensional* since there is no structure to the grid.

```
TYPE ObsGridType
  TYPE (DecompType)    :: Decomp          ! Decomposition
  INTEGER              :: Nobservations   ! Total points
END TYPE ObsGridType
```

The data corresponding to a this grid data structure is a set of vectors, one for the observation values and several for *attributes* of those values, e.g., the latitude, longitude and level at which an observation was taken, etc.

6.3 Finite Element Grid

This grid is a possible extension for later needs and would support a decomposed finite element grid. Currently only finite difference methods are employed in the DAO's software, but improvements could very well include finite element methods which support the irregular nature of many portions of atmospheric science codes.

```
TYPE FinElType
  TYPE (DecompType)    :: DecompVertices ! Vertex decomposition
  TYPE (DecompType)    :: DecompElements ! Element decomposition
```

```
  INTEGER                 :: Nvertices      ! Global number vertices
  INTEGER                 :: Nelements      ! Global number elements
END TYPE FinElType
```

The distribution of the finite element grid implies not only a decomposition of the vertices, but also of the elements. As is the case of other grids, the actual data is not carried in the grid data structure but rather in arrays whose distribution is dictated by the decompositions. For example, a vector with the decomposition **DecompElements** with length N_{local} elements will contain the constituent vertices for each element. There will be several vectors with decomposition **DecompVertices** of length N_{local} vertices which contain the coordinates of the vertices, as well one or more such vectors to describe the physical quantities sought in the vertices.

7 Examples and Results

Several examples of the PILGRIM library are presented here. These arise from the GEOS DAS application, generally in the interfaces between different code sections which work on different grids. It is important to remember that the two grids involved do not need to have the same distribution or the number of grid points. There are several different grids in use in GEOS DAS at run-time and correspondingly several different transformations between pairs of these grids.

7.1 Grid Rotation and Stretching

An example of a non-trivial transformation employed in atmospheric science applications is grid rotation [20]. Computational instabilities from finite difference schemes can arise in the polar regions of the geophysical grid when a strong cross-polar flow occurs. By placing the pole of the computational grid to the geographic equator, however, the instability near the geographic pole is removed due to the vanishing Coriolis term.

It is generally accepted that the physical processes, such as those related to long- and short-wave radiation can be calculated directly on the geophysical grid. Dynamical processes, where the numerical instability occurs, need to be calculated on the computational grid. An additional refinement is to calculate the dynamics on a rotated *stretched* grid, in which the grid-points are not uniform in latitude and longitude. To reiterate, the **LatLonGridType** allows for both variable lat-lon coordinates as well as the three parameters (tilt, rotation, and precession) to describe any lat-lon view of the world when the poles are assigned to a new geographical location. The grid rotation (without stretching) is depicted in Figure 3.

The routine **GridTransform** defines a distributed sparse matrix based on a bi-linear or bi-cubic interpolation from the geophysical grid (**GeoPhysGrid** of type **LatLonGridType**) to the computational (**CompGrid** of type **LatLonGridType**). A similar call is needed to create the reverse transform from the computational grid back to the geophysical.

```
GridTransform( GeoPhysGrid, CompGrid, MatrixGeoToComp )
GridTransform( CompGrid, GeoPhysGrid, MatrixCompToGeo )
```

It would be natural to use the same decomposition for both the geophysical and computational grids. It turns out, however, that this ignores data locality inherent to this transformation (see Figure 4). If the application could have unlimited freedom to chose the decomposition of the computational grid,

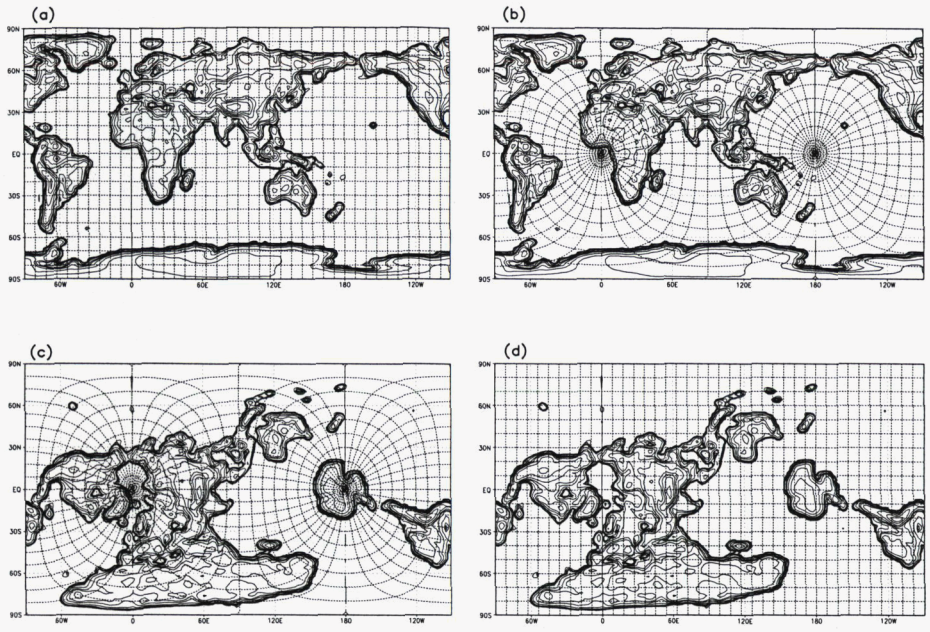

Figure 3: The use of the latitude-longitude grid (a) and (c) as the computational grid results in instabilities at the poles due to the Coriolis term. The instabilities vanish with on a grid (b) where the pole has been rotated to the equator. The computational grid is therefore a lat-lon grid (d) where the "poles" on the top and bottom are in the Pacific and Atlantic Oceans, respectively.

the forward and reverse grid rotations would have excellent data locality, and the matrix application would be much more efficient.[1] Unfortunately, practicality limits the decomposition of both the geophysical and computational grids to be a "checkerboard" decomposition of the horizontal cross-section.

However, there is still several degrees of freedom in the decomposition, namely the number of points on each PE and the assignment of local regions to PEs. While an approximately uniform number of points per PE is generally best for the dynamics calculation, the assignment of PEs is arbitrary. The following optimization is therefore applied: before the grid rotation, the potential communication pattern of a naive computational grid decomposition is analyzed by adopting the decomposition of the geophysical grid. With a heuristic method, this analysis leads to a *permutation* of PEs for the computational grid which reduces communication (Figure 4). After this the decomposition of the computational grid is defined as a permuted version of the geophysical grid. An outline of the code is as given in Algorithm 1

Algorithm 1 (Grid Permutation) *Given the geophysical grid decomposition, find a permutation of the PEs which will maximize the data locality of the geophysical-to-computational grid transformation, create and permute the computation grid decomposition, and define the transformation in both directions.*

[1] The reader will note that a simply connected region in one domain will map to at most two simply connected regions in the other

Unpermuted Communication Matrix:

$$\begin{bmatrix} 0 & 219 & 5905 & 172 & 0 & 97 & 507 & 12 \\ 53 & 5731 & 690 & 2 & 1 & 303 & 132 & 0 \\ 3 & 477 & 136 & 0 & 53 & 5727 & 516 & 0 \\ 0 & 97 & 335 & 4 & 3 & 366 & 5942 & 165 \\ 516 & 0 & 53 & 5727 & 136 & 0 & 3 & 477 \\ 5967 & 166 & 2 & 341 & 371 & 4 & 0 & 61 \\ 543 & 12 & 0 & 61 & 5941 & 172 & 0 & 183 \\ 133 & 0 & 1 & 302 & 760 & 1 & 54 & 5661 \end{bmatrix}$$

Permuted Communication Matrix:

$$\begin{bmatrix} 5967 & 166 & 2 & 341 & 371 & 4 & 0 & 61 \\ 53 & 5731 & 690 & 2 & 1 & 303 & 132 & 0 \\ 0 & 219 & 5905 & 172 & 0 & 97 & 507 & 12 \\ 516 & 0 & 53 & 5727 & 136 & 0 & 3 & 477 \\ 543 & 12 & 0 & 61 & 5941 & 172 & 0 & 183 \\ 3 & 477 & 136 & 0 & 53 & 5727 & 516 & 0 \\ 0 & 97 & 335 & 4 & 3 & 366 & 5942 & 165 \\ 133 & 0 & 1 & 302 & 760 & 1 & 54 & 5661 \end{bmatrix}$$

Figure 4: The above matrices represent the number of vector entries requested by a PE (column index) from another PE (row index) to perform a grid rotation for one 72×48 horizontal plane (i.e., on matrix-vector multiplication) on a total of eight PEs. The unpermuted communication matrix reflects the naive use of the geophysical grid decomposition and PE assignment for the computational grid. The permuted communication matrix uses the same decomposition except the assignment of local regions to PEs is permutated. The diagonal entries indicate data which can be fetched from the local PE without message-passing and represent work which can be overlapped with the asynchronous communication involved in fetching the non-local data. The diagonal dominance of the communication matrix on the right translates into a considerable performance improvement.

```
SparseMatrixCreate( ..., MatrixGeoToComp )
SparseMatrixCreate( ..., MatrixCompToGeo )
DecompCreate( ..., GeoPhysDecomp )
LatLonCreate( GeoPhysDecomp, ...., GeoPhysGrid )
AnalyzeGridTransform( GeoPhysDecomp, ...., Permutation )
DecompCopy( GeoPhysDecomp, CompDecomp )
DecompPermute( Permutation, CompDecomp )
LatLonCreate( CompDecomp, ...., CompGrid )
GridTransform( GeoPhysGrid, CompGrid, MatrixGeoToComp )
GridTransform( CompGrid, GeoPhysGrid, MatrixCompToGeo )
```

In `GridTransform` the coordinates of the original lat-lon grid are mapped to a new lat-lon grid. Interpolation coefficients are determined by the proximity of rotated grid points of one grid to grid points on the other grid (see Figure 3). Various interpolation schemes can be employed, e.g., bi-linear or bi-cubic; the latter is employed in GEOS DAS. Clearly the transformation matrix can be completely defined by the two lat-lon grids — the values on those grids are not necessary.

Once the transformation matrix is defined, sets of grid values, e.g., individual levels or planes of atmospheric data, can be transformed ad infinitum using a matrix-vector (or matrix-matrix) multiplication.

```
DO L = 1, GLOBAL_Z_DIM
  CALL SparseMatVecMult ( MatrixGeoToComp,
&                         1.0, In(1,1,L), 0.0, Out1(1,1,L) )
```

```
END DO
```

Alternatively, it the transformation of the entire 3-D grid can be performed in one swoop with a call to the matrix-matrix product.

```
CALL SparseMatMatMult ( MatrixGeoToComp, LOCAL_X_DIM*LOCAL_Y_DIM,
&                       GLOBAL_Z_DIM, 1.0, In, 0.0, Out2 )
```

Note that the pole rotation is trivial (embarrassingly parallel) if any given plane resides entirely on one PE, i.e., if the 3-D array is decomposed in the z-dimension. Unfortunately, there are compelling reasons to distribute the data in vertical columns with the checkerboard decomposition.

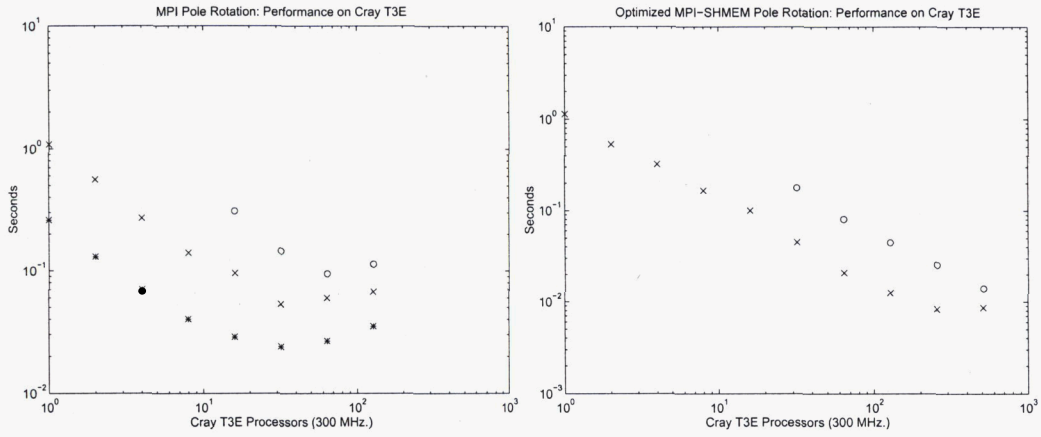

Figure 5: With a naive decomposition of both the geophysical and computational grids and a straightforward MPI implementation, the performances at the left for the $72 \times 46 \times 70$ (*), $144 \times 91 \times 70$ (x), and $288 \times 181 \times 70$ (o) resolutions yield good scalability only to some 10-50 processors. The optimized MPI-SHMEM hybrid version on the right scales to nearly the entire extent of the machine (512 processors).

Figure 5 compares the performance of the naive, non-permuted rotation with that of the permuted rotation on the Cray T3E. In the latter case, a further optimization is performed in that the non-blocking MPI primitives used in `ParBeginTransfer` are replaced by faster Cray SHMEM primitives. The result of these optimization is the improvement in scalability from tens of PEs to hundreds of PEs. The absolute performance in GFlop/s is presented in Figure 6.

7.2 Interpolation Lat-lon to Observation Grid

In GEOS DAS, observationaly data is incorporated into the time evolution of the GCM with the PSAS package. Every six hours of assimilation, approximately 80,000 *observations* are read in which contain values and associated attributes, such as latitude, longitude and level. Currently the observations are replicated on all PEs, although in the future they may be distributed over the observation grid described in Section 6.2. The parallelization of this portion of the *observer* is considered here.

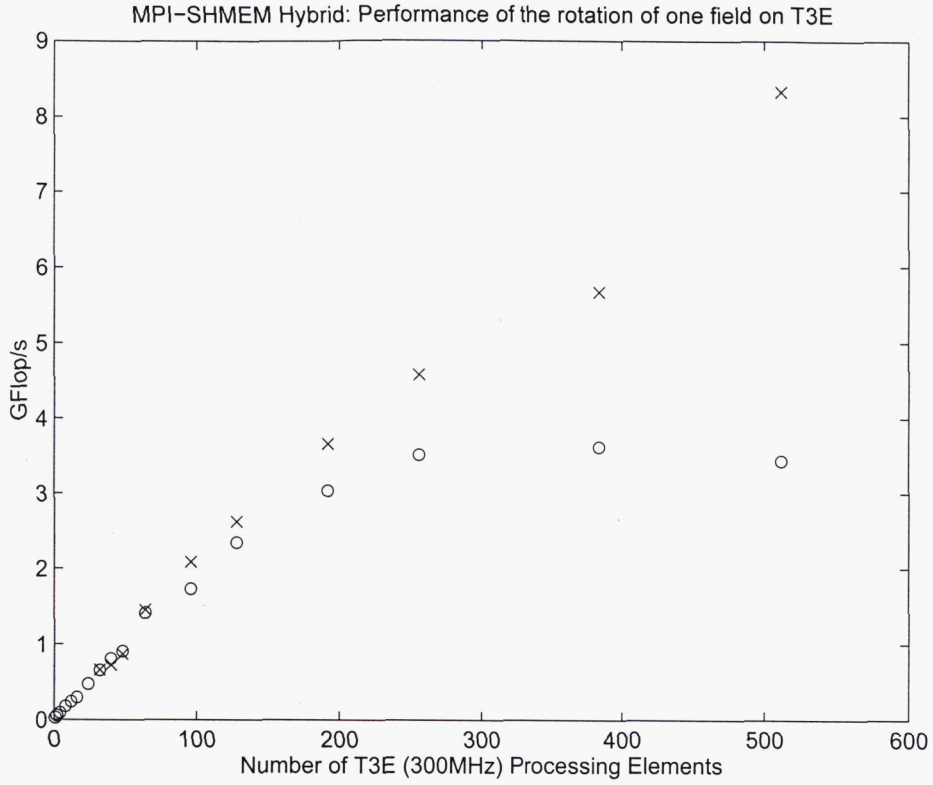

Figure 6: The GFlop/s performances of the grid rotation on grids with $144 \times 91 \times 70$ (o), and $288 \times 181 \times 70$ (x) resolutions is depicted. These results are an indication that the grid rotation will not represent a bottleneck for the overall GEOS DAS system.

The *forecast* of the former must be interpolated to the grid of observations, on which the latter then operates. This is a linear transformation which depends only on the latitude, longitude and level attributes. Thus at every analysis step, the following interpolation operation must be performed

$$y = Ax \quad \text{where} \quad \begin{aligned} A &\in \mathcal{R}^{n_{obs} \times n_{fcst}} \\ x &\in \mathcal{R}^{n_{fcst}} \\ y &\in \mathcal{R}^{n_{obs}} \end{aligned}$$

The interpolation is 3-D bi-linear and requires only the eight points surrounding a given observation (perhaps fewer if some of the forecast grid values are undefined). The number of non-zeros in A is therefore 640,000, making the problem tractable to solve with the sparse linear algebra utilities. The key question is whether the overhead of constructing a distributed matrix is worthwhile for the small number of transformations (8) at every analysis step. The answer seems to be yes if the matrix entries are inserting in parallel using the `SparseInsertLocalEntries` primitive and if the local indices required by it can be determined in an efficient way. Indeed, the decomposition of the lat-lon grid is simply the 2-D block-block decomposition of the cross-section, and therefore determining local indices is efficient.[2] The matrix row index corresponds to the local index of the

[2] Look-up tables further improve performance

distributed observation vector and does not require a conversion from a global index.

The routine to define the transformation matrix takes arrays of length $N_{localobs}$ for the the latitudes, longitudes and levels, and, given the geophysical grid decomposition, returns the transform matrix:

```
VertInt( N_localobs, Lats, Lons, Levels, GeoDecomp, MatrixGeoToObs )
```

Algorithm 2 contains a summary of this routine's internal structure.

Algorithm 2 (Define Geophysical to Observation Grid Transformation) *Given the attributes latitude, longitude, levels, and the decomposition of the geophysical grid, define a distributed sparse matrix to describe the linear transformation from the geophysical grid to the observation locations.*

For all i over all local observations:

1. *Using the geophysical grid decomposition find the local indices and PE assignments of the eight neighbors surrounding observation i as defined by its latitude, longitude and level.*
2. *Take into account whether any of the neighbors is an undefined geophysical point.*
3. *Construct the coefficients for the eight (or fewer, if some are undefined) neighbors.*
4. *Input the i^{th} row of the matrix with* `SparseInsertLocalEntries`

The actual linear transformation then becomes a straightforward application of the sparse linear algebra utilities[3]:

```
SparseMatVecMult( MatrixGeoToObs, 1.0, Forecast, 0.0, ObsValue )
```

8 Summary

We have introduced the parallel grid manipulations needed by GEOS DAS and the proposed PILGRIM library designed to implement them. PILGRIM is modular and extensible, allowing us to support various types of grid manipulations and perform with high performance and scalability onlarge on state-of-the-art parallel computers with a large number (> 100) of processors.

Results from the grid rotation problem were presented, and they indicate that PILGRIM will perform adequately even on the most communication-bound grid manipulations needed in our applications. We are hoping to extend the usage of PILGRIM in GEOS DAS to the interface between the forecast model and the statistical analysis, to perform further optimizations on the library, and to offer the library to the public domain in the future.

Acknowledgments

A first technical review of PILGRIM was conducted on May 15, 1997 at the Data Assimilation Office. We would like to thank Jay Larson (review leader), Max Suarez, and Dan Schaffer for their

[3] Some post-processing must be performed to locate and set observation values which are undefined.

valuable suggestions. The work of Will Sawyer and Peter Lyster at the Data Assimilation Office was funded by the High Performance Computing and Communications Initiative (HPCC) Earth and Space Science (ESS) program.

References

[1] G. Heiser, C. Pommerell, J. Weis, and W. Fichtner. Three dimensional numerical semiconductor device simulation: Algorithms, architectures, results. *IEEE Transactions on Computer-Aided Design of Integrated Circuits*, 10(10):1218–1230, October 1991.

[2] A. Ecer, J. Hauser, P. Leca, and J. Périaux. *Parallel Computational Fluid Dynamics.* Elsevier Science Publishers B.V. (North–Holland), Amsterdam, 1995.

[3] H.-P. Kersken, B. Fritzsch, O. Schenk, W. Hiller, J. Behrens, and E. Kraube. Parallelization of large scale ocean models by data decomposition. *Lecture Notes in Computer Science*, 796:323–336, 1994.

[4] Patrick Knupp and Stanly Steinberg. *Fundamentals of Grid Generation.* CRC Press, Boca Raton, FL, 1994.

[5] M. T. Jones and P. E. Plassmann. Parallel algorithms for the adaptive refinement and partitioning of unstructured meshes. In IEEE, editor, *Proceedings of the Scalable High-Performance Computing Conference, May 23–25, 1994, Knoxville, Tennessee*, pages 478–485, 1109 Spring Street, Suite 300, Silver Spring, MD 20910, USA, 1994. IEEE Computer Society Press.

[6] W. L. Briggs. *A Multigrid Tutorial.* SIAM, 1987.

[7] S. T. Barnard and H. D. Simon. A Fast Multilevel Implementation of Recursive Spectral Bisection for Partitioning Unstructured Problems. Technical Report RNR-092-033, NASA Ames Research Center, Moffett Field, CA 94035, November 1992.

[8] L. L. Takacs, A. Molod, and T. Wang. Documentation of the Goddard Earth Observing System (GEOS) General Circulation Model — Version 1. Technical Memorandum 104606, NASA, Code 910.3 Greenbelt MD, 20771, USA, 1994.

[9] A. da Silva and J. Guo. Documentation of the Physical-space Statistical Analysis System (PSAS), Part 1: The Conjugate Gradient Solver, Version PSAS 1.00. DAO Office Note 96-02, Data Assimilation Office, NASA, Code 910.3 Greenbelt MD, 20771, USA, 1996.

[10] C. L. Lawson, R. J. Hanson, D. R. Kincaid, and F. T. Krogh. Basic Linear Algebra Subprograms for Fortran Usage. *Transactions on Mathematical Software*, 5(3):308–323, September 1979.

[11] Timothy Budd. *Object-Oriented Programming.* Addison-Wesley, New York, N.Y., 1991.

[12] MPIF (Message Passing Interface Forum). MPI: A Message-Passing Interface Standard. *International Journal of Supercomputer Applications*, 8(3&4):157–416, 1994.

[13] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: A Users' Guide and Tutorial for Networked Parallel Computing.* MIT Press, 1994.

[14] Cray Research. CRAY T3E Applications Programming. Software Education Services.

[15] DAO staff. Algorithm Theoretical Basis Document. NASA Technical Memorandum, Data Assimilation Office, NASA, Code 910.3 Greenbelt MD, 20771, USA, 1996.

[16] DAO Staff. GEOS-3 Primary System Requirements Document. `http://dao/Intranet/CM/Library/Requirements/GEOS-3/PriRqmts.html`, 1996.

[17] L. C. McInnes and B. F. Smith. Petsc 2.0: A case study of using mpi to develop numerical software libraries. In *1995 MPI Developers' Conference*, June 1995.

[18] S. A. Hutchinson, J. A. Shadid, and R.S. Tuminaro. *The Aztec User's Guide - Version 1.0.* 1995.

[19] Oliver Bröker, Vaibhav Deshpande, Peter Messmer, and William Sawyer. Parallel library for unstructured mesh problems. Tech. Report CSCS-TR-96-15, Centro Svizzero di Calcolo Scientifico, CH-6928 Manno, Switzerland, May 1996.

[20] M. J. Suarez and L. L. Takacs. Documentation of the ARIES/GEOS Dynamical Core: Version 2. NASA Technical Memorandum 104606, NASA, Code 910.3 Greenbelt MD, 20771, USA, 1995.