

Storing and Manipulating Gridded Data in Databases

Barrodale Computing Services Ltd. (BCS)¹

Abstract

Gridded data is fundamental to many scientific and technological applications, ranging from meteorology and oceanography to petroleum exploration and extraction. The purpose of this short article is to provide a review of features and applications of gridded data, to explain the advantages that a modern database can provide when storing and manipulating gridded data, and to highlight the need for some additional functionality that is critical to a gridded database's ability to perform its tasks satisfactorily. We then describe a software module (the BCS Grid DataBlade) that plugs into a database server to provide this functionality, thereby enabling users to *efficiently* and *rapidly* store and update gridded data and retrieve information.

¹ BCS welcomes inquiries via BCSinfo@barrodale.com; see also <http://www.barrodale.com>.

Table of Contents

Introduction.....	1
Overview of Gridded Data.....	1
Grid Concepts	1
Grids in Modeling Applications.....	3
Grids in Data Analysis	4
Grid Storage Formats.....	5
The Role of Databases for Gridded Data.....	5
Implementing a Gridded Database.....	7
The BCS Grid DataBlade.....	8
Appendix - SQL Datatypes and Functions	10
Datatypes.....	10
Functions.....	10
Example Uses.....	11

Introduction

The amount of data that will be recorded in the next 3–5 years will probably exceed all of history's recorded data up to the present time, and a significant portion of this data will occur in the form of grids. Gridded data occurs in many specialized application areas such as meteorology, oceanography, hydrology, surveying, civil engineering, astronomy, non-destructive testing, medical imaging, social sciences, and exploration systems for oil, natural gas, coal, and diamonds. These data sets range from simple, uniformly spaced grid points along a single dimension to multidimensional grids containing several different types of grid values. For example, if ocean measurements (temperature, salinity, pressure) were recorded every hour at spacings every one meter in depth and every ten meters in two horizontal dimensions, this would result in a 4D grid having three spatial dimensions and one temporal dimension, with three values attached to each grid point.

In the past, grids were typically stored in simple files and then manipulated by programs that operated on these files. Nowadays, there is increasing justification for using database management systems (DBMSs) to store and manipulate gridded data, with applications often being built around integrated databases. The principal advantages of such databases are their ability to (i) ensure data integrity and consistency, and (ii) provide diverse users with independent and effective access to this data across multiple applications and systems.

However, implementing an *efficient* gridded DBMS can be very challenging, particularly when it involves Binary Large Objects (BLOBs), user-defined datatypes (UDTs) that encapsulate grid data structures and attributes, and user-defined routines (UDRs) with which applications can create, manipulate and access the gridded data stored in these new datatypes.

BCS has recently developed a very efficient technology that supports database storage, update, and fast retrieval of gridded data; it uses BLOBs, UDTs and UDRs, and its main features are described in the last section of this article. Our first implementation of this technology is as a Grid DataBlade² for use with the IBM/Informix Dynamic Server (IDS) DBMS; an analogous Grid Extender is now under development at BCS for use with the IBM DBMS product DB2.

Overview of Gridded Data

Grid Concepts

Some examples of gridded data of different dimensions are as follows:

² An IBM/Informix DataBlade is a collection of datatypes, their associated functions and operators, and access methods. Oracle uses the term Cartridge to describe a similar concept, and IBM uses the term Extender for DB2.

- 1D: simple time series (e.g., stock market indices)
- 2D: raster images, or numerical solutions to partial differential equations of two variables
- 3D: spatial volumes, or images at various time points
- 4D: volumes at various time points (e.g., dimensions of latitude, longitude, depth, and time)
- “5D”: 4D grids with values for a set of (possibly unrelated) variables at each 4D grid point³.

Many applications use a simple type of grid, with orthogonal axes, evenly spaced grid points, and a single value at each grid point. However, many special cases also exist where more complicated grids must be represented and manipulated. These include:

- ***Irregularly spaced grids.*** In this case the sampling points along one or more dimensions are not evenly spaced. Thus, along an altitude axis, say, certain data could be recorded at heights of 50, 100, 200, 500, 1000, 2000, 3000, 4000 and 5000 m. Applications that use a grid of this nature must obviously take uneven spacing into account.
- ***Grids with multiple values*** (including vector values). In many applications (e.g., meteorology) it is appropriate to have values for several different physical parameters at each grid point. This raises the issue of whether to store them in the form of a single grid with multiple values, or to use multiple grids, each with a single value.
- ***Grids with missing values.*** Certain grids might not be populated with values at all grid points (e.g., oceanographic forecasts of wave height masked with missing values to exclude grid points on land). Applications using such grids must take these missing values into account.

Operations on grids can sometimes be simple, such as extracting the value at a particular grid point. However, in many applications more complicated and sophisticated operations on the grid are required to generate the output of interest (i.e., desired information in the form of a data product). Typical examples are as follows:

- ***Interpolation:*** estimates the value of a parameter for grid positions not corresponding exactly with a grid point.
- ***Affine transformation:*** translates, scales, and/or rotates the grid in order to perform the required computations.
- ***Projection:*** obtains a grid involving geographic coordinates in a different map projection from the one used in generating and storing the grid.

³ This “5D” example is not, strictly speaking, a grid in the conventional sense. However, this 5D designation is consistent with current usage in the technical literature, and the BCS Grid DataBlade handles this structure.

- **Windowing and subspace generation:** optimizes the extraction of data products involving just a very small portion of the overall grid. These include “sticks” (1D vectors of values), slices (2D planar portions), and sub-volumes.
- **Convenient update:** selectively updates portions of the grid without having to rewrite the entire grid.

Clearly, users want to be able to handle these special grids and generate these types of data products without undue effort (e.g., without having to extract the entire grid from the database).

Grids in Modeling Applications

Broadly speaking, gridded data arises in two main areas of application: modeling applications (often involving the solution of differential equations or difference equations) and data analysis applications (which are discussed in the next section). Examples of modeling applications in which grids play a key role are as follows:

Meteorology. Grids of meteorological data are central to the prediction of weather on both local and global scales. While satellite and other forms of imagery provide information on the current weather, these images are snapshots of the present and past, whereas grids provide the predictive power needed for weather-related decisions about the days ahead. Typically, 3D grids are populated with estimated values of physical parameters such as temperature, pressure, wind speed, and relative humidity, and the evolution of these values over time is predicted by solving sets of difference equations. The result is a 4D space-time grid of predicted values of the physical parameters. Such grids, generated over various scales, are often sub-sampled and interpolated and the results are then projected for display and interpretation. Many weather-related data products may also be generated from the information in these grids. Major producers of meteorological grids are the civilian and military weather forecasting agencies, including NOAA (<http://www.noaa.gov/>) and FNMOC (<http://www.fnoc.navy.mil/>) in the USA, and the MET Office in the UK (<http://www.met-office.gov.uk/>).

Oceanography. Grids are used in modeling ocean circulation and ocean-atmosphere interaction. These applications are similar to meteorological applications but also take the shorelines and subsurface ocean physical parameters into account in the modeling, and typically have longer time horizons than for weather prediction.

Climatic modeling. Global climate modeling is often based on solving differential equations for grids encompassing the Earth’s surface. In this case, the time horizons are often decades or centuries. (See <http://www.arsc.edu/pubs/challenges/v8n2/ClimateChange.html>).

Fluid dynamics. Certain other types of mathematical modeling are carried out on grids, and fluid dynamics is just one example. The motion of a fluid in response to a forcing function can be modeled using finite element analysis applied to a 3D grid.

Other specific examples of modeling applications involving grids are hurricane analysis, optimal extraction of mine resources and petroleum reservoirs, air quality control strategies, and forest fire burn area predictions.

Grids in Data Analysis

Non-Destructive Testing. Grids of 3D data are increasingly used in the nondestructive testing of materials using ultrasonic inspection techniques. These are particularly vital to the aviation industry for inspection of aircraft components to detect defects and delaminations. In these applications, a transducer is moved over the surface in a raster pattern and an ultrasonic pressure trace is recorded at each position. Hence, the three dimensions are x , y , and t . These data sets may be processed in various ways to enhance their resolution, and features of interest may be analyzed using special-purpose visualization tools (<http://www.barrodale.com/nde/index.htm>). Also, gridded data collected during one inspection may be compared with data obtained for the same component during an earlier inspection, to detect any significant changes.

Geophysics. Prospecting for natural resources such as oil and gas involves the acquisition, analysis and interpretation of gravimetric, electromagnetic, and seismic data stored as large 2D, 3D and, sometimes, 4D grids. Here, the data collected is extensively processed both before and after grid formation. These grids are collected over large areas, both on land and over the ocean floor. A common application is to generate 2D slices along an arbitrary vertical plane (cross-sections) through this data. It is also possible to combine several time-separated data sets and to model the changes within the volume (e.g., resulting from petroleum removal). In addition to oil and gas prospecting, other areas of geophysics that involve grids include mineral prospecting (e.g., diamonds), geothermal analysis, and plate tectonics modeling.

Medical Imaging. Various medical imaging techniques have recently come into existence that can produce 3D (and occasionally 4D) gridded data. These include ultrasonic pulse echography, computed axial tomography (CAT), magnetic resonance imaging (MRI), and positron emission tomography (PET). The ability of our grid manipulation technology to retrieve *oblique* 2D slices from 3D data sets is demonstrated at http://www.barrodale.com/grid_Demo/index.html (the first demo provided); here we utilize a 1.6GB 3D grid from a Visible Human Project file system containing 1871 parallel raster images spaced at 1mm intervals of a cryosectioned male subject.

Simulated Surgery. Another recent application of gridded data is in simulated surgery. In this application, a 3D model of the body region subject to surgery is obtained through imaging techniques, and the effects of a given surgical scenario are assessed before the surgery actually

proceeds. This allows the refinement of the proposed procedures to produce the most desirable result (e.g., in facial reconstruction).

Other specific examples of data analysis applications based heavily on grids are aerial site mapping, linear corridor identification, watershed delineation, hydrographic surveying, highway engineering, and demographic predictions.

Grid Storage Formats

Because of the widespread use of grids in many diverse applications, a number of different standards and formats have been defined for storing them. Some of these are:

- CDF (Common Data Format) is a library and toolkit for storing, manipulating, and accessing multidimensional data sets. The basic component of CDF is a software interface that is a device independent view of the CDF data model.
- GRIB (GRIdded Binary) is the World Meteorological Organization (WMO) standard for gridded meteorological data.
- HDF (Hierarchical Data Format) is a self-defining file format for transfer of various types of data between different machines. The HDF library contains interfaces for storing and retrieving compressed or uncompressed raster images with palettes, and an interface for storing and retrieving n-dimensional scientific data sets together with information about the data, such as labels, units, formats, and scales for all dimensions.
- NetCDF (Network Common Data Form) is an interface for scientific data access that implements a machine-independent, self-describing, extendible file format.
- SDTS (Spatial Data Transfer Standard) is a Federal standard (Federal Information Processing Standard (FIPS) 173) for transfer of geologic and other spatial data.

The Role of Databases for Gridded Data

Until recently, gridded data was usually stored in files and users ran applications on these files to generate the required data products. However, there is now increasing recognition that it is often advantageous to store gridded data in a modern database and allow users to extract data products by running queries against the database. Three advantages are as follows:

- ***Uniform treatment of data items.*** By storing gridded data together with its metadata and other data, applications can use the simple SQL interface to perform complex queries based on any of these data items, including dynamically-derived properties of the gridded data. An

example of this would be the following pseudo-SQL query, which returns all pressures at a depth of 100 m, collected by a particular device:

```
select GRDExtract(gridColumn, "extraction_specification")
   from tablename
  where inside(100, depth_range)
     and collection_device = "pressure_transducer";
```

This basic query could be extended by modifying the text of the extraction specification to support more refined requests, such as selecting pressures collected during the last five hours, converting to a Lambert conformal projection, etc. In contrast, gathering this information when the gridded data is on a file system would involve a complex application of not only queries but also code to locate, transform and subset the relevant data.

- ***Client-server issues.*** In a traditional database or file-based environment, custom functionality is implemented purely on the client side, resulting in “fat clients”. Modern DBMSs, in addition to allowing UDTs, also allow UDRs to be defined in languages such as C or Java, which provide much greater control over where functionality resides. Data can be assembled, processed and/or disassembled on either the client or the server, or both, with attendant tradeoffs.
- ***Synchronized concurrent access to data.*** Multiple users of the database can safely query the same data simultaneously. With a file-based approach, there is a danger that the update activity of one application might result in another application seeing inconsistent data.

However, in order to realize the true potential of gridded databases, it is also necessary to provide for certain critical functionality to be available as an integrated component of the overall system. Some examples of sophisticated operations were provided in the *Overview of Gridded Data* section above.

Expanding on one of these examples, it is our experience that data products required by users of large grids often involve only very small portions of the grid. For instance, on a global weather grid, the region of interest may only be a few tens of kilometers on a side; on an oceanographic data grid, the required data might be a 2D slice 100 km long. Under these conditions, only a tiny fraction of the grid needs to be used to generate the products. Certain DBMSs can be made to perform such manipulations very efficiently through the use of tiling combined with BLOBs. By taking this approach, only those tiles that contain the data of interest are moved into memory during data product generation, and the vast majority of the gridded data is not involved in the data transfer or the computations. Understandably, this can be a very important performance consideration in the case of Web-based applications.

As already noted, only certain types of DBMSs are suitable candidates for achieving all the potential advantages of using a database to store gridded data. In particular, object-relational DBMSs provide the support for UDTs and UDRs required to realize these advantages.

Implementing a Gridded Database

Once a decision is made to store gridded data sets in a DBMS (perhaps based on considerations such as those in the previous section), users must choose an implementation method. There are two relatively straightforward approaches that can be taken; however, each may have severe performance penalties.

In the first approach, the value (or set of values) of each grid point is stored in a separate row, along with fields that describe the position of the grid point. For instance, a 4D grid of floating-point values might be represented by the following SQL command:

```
Create table A4Dgrid (  
    x integer,  
    y integer,  
    z integer,  
    t integer,  
    GridPointValue float);
```

The most obvious drawback to this approach is that extra space is taken by the x , y , z and t fields used to record the position of each grid point. The less obvious drawback, and more serious one, is the need for a large multidimensional index to be able to find required rows in the table in an efficient manner. The space taken by a multidimensional index will be larger than the size of the table itself if all x , y , z , and t are used as the key.

In the second approach, a multidimensional array is simply written to a BLOB, which is then stored in a field of a table in the database. Applications fetch the contents of the BLOB when they wish to operate on the data. The main drawback to this approach is that it either requires the entire grid to be passed to the client, or it requires that the client perform a large number of BLOB input/output operations to read just the portions of the grid that are required. Both of these possibilities are very demanding on the network connecting the client to the server, and this approach also requires a great deal of logic on the client side.

In contrast, a more efficient solution developed at BCS combines both approaches. We store the grid data inside BLOBs, and embed the grid manipulation logic inside the server as UDRs; this removes the need to record explicitly the position of each grid point, it does not require the use of multidimensional indexes, and it reduces the amount of data transfer from the server to the client to just that portion which is required for the application. With the addition of UDTs, additional metadata (such as coordinate system information, dimension descriptions, grid value type description, etc.) can be added to make the grid self-describing to database clients.

In short, the technology described in the next section was designed to overcome many of the inconveniences and inefficiencies that users can encounter when storing and manipulating gridded data in a database.

The BCS Grid DataBlade

BCS has recently developed a Grid DataBlade for storage of gridded data and efficient retrieval of data products. The Grid DataBlade runs under IBM/Informix Dynamic Server 9.x (IDS) and processes queries on the database server, thereby minimizing the amount of network input/output and client-side CPU time required. In addition, we utilize an internal tiling scheme, certain affine transformations, IDS SmartBLOB⁴ technology, and the Spatial DataBlade (supplied at no charge to customers with a current IDS license), to provide an effective and very efficient (order of magnitude speed-ups for some applications) environment for housing and accessing gridded data. Specifically, the Grid DataBlade:

- is designed to handle 4D (and “5D”) grids as a standard. Grids of lesser dimensionality are handled by taking one or more of the grid dimensions to be 1.
- stores grids using a tiling scheme in conjunction with SmartBLOBs, with user control over the tile size. This allows very efficient generation of data products that involve only a small portion of the data in the overall grid (e.g., extracting planar oblique slices from volumetric grids, or retrieving 1D probes or sticks from 3D or 4D grids).
- can store the data in, and convert it between, more than 40 different planar mapping projections supported by the IBM/Informix Spatial DataBlade (the second demo provided at http://www.barrodale.com/grid_Demo/index.html uses the Grid DataBlade and the Spatial DataBlade to resample and re-project a 386 MB raster image of the World).
- can handle irregularly spaced grids in any or all of the grid dimensions.
- can handle the presence of multiple vector and/or scalar values at each grid point.
- provides interpolation options using N-linear or nearest-neighbor schemes.
- provides for convenient loading and extraction of grid files to/from the database via a specific form of the commonly used NetCDF format, termed Grid Import-Export Format (GIEF).
- provides application programming interfaces for C, Java and SQL.
- often provides more than 50-fold increases in speed of data product generation compared to the conventional approach that does not involve tiling or SmartBLOBs.
- is supplied with full user/programmer documentation.

In summary, there is now a strong trend towards storing and manipulating gridded data in DBMSs, and although it might appear fairly straightforward to the first-time user to design and implement such a strategy, our own experience confirms that an efficient realization will involve a steep learning curve of many months’ duration. Nevertheless, the advantages of gridded databases (data integrity and consistency, independent and effective access to data by diverse

⁴ SmartBLOBs can store much larger amounts of data than traditional BLOBs, and it is possible to access and modify the content of a SmartBLOB without having to extract the entire BLOB from the database. This latter property can have a significant impact on the time required for data transfer.

users of multiple applications) are very appealing, and significant benefits can indeed be achieved through the use of modern DBMS technology (including BLOBS, UDRs and UDTs). The BCS Grid DataBlade allows users to experience these benefits with little effort or delay; as one satisfied customer reported, “It worked right out of the box.”

As was mentioned in the *Introduction*, an analogous Grid Extender is now under development at BCS for use with the IBM DBMS product DB2; it is our intention to also implement this grid technology with other DBMSs in due course.

For further information on the Grid DataBlade, and to see interactive demonstrations using the Grid DataBlade, please refer to <http://www.barrodale.com>.

Appendix - SQL Datatypes and Functions

Datatypes

The Grid DataBlade provides two new datatypes called GRDValue and GRDSpec:

- GRDValue stores 4D grids and their meta data.
- GRDSpec is used to specify how to create a new GRDValue from an existing GRDValue.

Functions

Examining a GRDValue

GRDResolution: returns the resolution along a specified grid dimension.

GRDSrid: returns the SRID of the projection system of the GRDValue.

GRDToGeodetic: returns a bounding box that can be spatially indexed.

GRDStartPoint: returns the location of the first grid point.

Modifying a GRDValue

GRDExtend: enlarges the size of GRDValue.

GRDUpdate: copies the contents of one grid into another existing grid.

Extracting a GRDValue

GRDExtract: applies a GRDSpec to an existing GRDValue to produce a new GRDValue.

GIEF Support Functions

GRDFromGief: loads a GIEF/NetCDF file into a new row in a table of the database.

GRDRowToGief: saves a specified row of a table to a GIEF file, after applying a GRDSpec.

Additional Support Functions

GRDTrim: removes unneeded spaces from spatial reference text.

GRDGenSrid: insures the presence of an entry in the spatial references table for a given spatial reference text.

GRDIWholeSphere: determines whether a GRDValue covers the entire Earth.

GRDSecsToTime: returns the textual representation of the datetime represented by a time in seconds since 1970.

Example Uses

Loading a GIEF File into a Table

```
execute procedure GRDFromGief('myGiefFile.nc', 'myGridTable');
```

Copying the Contents of one Grid into another Grid

```
update myGridTable set gridColumn = (select unique gridColumn from myOtherGridTable);
```

Extracting a Subset of a Grid

```
select GRDExtract(gridColumn, '((dim_sizes 10 10 10 5)(translation 4 5 3 9))'::GRDSpec)
from myGridTable;
```

Extracting a Subset of a Grid into a File

```
select GRDRowToGief('myOutputFile.nc', 'myGridTable', rowid,
    '((dim_sizes 10 10 10 5)(translation 4 5 3 9))'::GRDSpec)
from myGridTable;
```