

Managing and Extracting Gridded Data: *Flat Files or BCS Database Extensions?*

January 2008

Executive Summary

This report compares the direct use of flat files (in netCDF format) with extractions from a database (PostgreSQL, in this case) containing the same file information. By arranging a chosen 4D dataset in tiles and storing them in the database, we show that PostgreSQL – **enhanced with either of two BCS database extension products (DBXten or the Grid Extension)** – offers performance that is comparable or superior to that obtained from a C program reading directly from the flat files. The 4D dataset used in the timing tests was from an ocean circulation model hindcast for the Indian Ocean and South China Sea.

For readers who prefer to postpone study of the test details, the table below provides a summary of our results. The times shown are in seconds per extraction; in each row the fastest time is in **boldface (green)**, the next fastest is *italicized (blue)* and the slowest time is in normal font (**red**). In short, apart from the tests involving latitude-longitude slices, where the netCDF file format lends itself to fast extractions, **our server extensions allow the database to perform comparably or better than direct manipulation of the netCDF files**. Here, DBXten executes faster than the Grid Extension; the latter product includes more far functionality for manipulating gridded data than does DBXten, and so it incurs somewhat greater overhead.

| Nature of Extraction | Extraction Size | DBXten Tile size is 1x16x22x37 | Grid Extension Tile size is 1x6x22x37 | netCDF Flat Files |
|--------------------------|-----------------|--------------------------------------|---|-------------------|
| Time Profile | 81x1x1x1 | 1.03 sec | <i>2.24 sec</i> | <i>1.30 sec</i> |
| Depth Profile | 1x66x1x1 | 0.07 sec | <i>0.17 sec</i> | <i>0.72 sec</i> |
| Latitude-Longitude slice | 1x1x50x50 | <i>0.06 sec</i> | <i>0.09 sec</i> | 0.03 sec |
| | 1x1x100x100 | <i>0.09 sec</i> | <i>0.13 sec</i> | 0.03 sec |
| | 1x1x200x200 | <i>0.17 sec</i> | <i>0.23 sec</i> | 0.04 sec |
| Depth-Latitude slice | 1x33x25x1 | 0.07 sec | <i>0.16 sec</i> | <i>0.37 sec</i> |
| | 1x33x50x1 | 0.09 sec | <i>0.20 sec</i> | <i>0.38 sec</i> |
| | 1x66x100x1 | 0.14 sec | <i>0.38 sec</i> | <i>0.80 sec</i> |
| | 1x66x200x1 | 0.20 sec | <i>0.59 sec</i> | <i>0.68 sec</i> |
| 4D Volume | 10x5x20x40 | 0.19 sec | <i>0.46 sec</i> | <i>0.69 sec</i> |
| | 10x10x40x80 | 0.34 sec | <i>0.99 sec</i> | <i>1.24 sec</i> |
| | 10x20x80x160 | 1.12 sec | <i>3.75 sec</i> | <i>2.49 sec</i> |
| Average Time | | 0.30 sec | <i>0.78 sec</i> | <i>0.73 sec</i> |

Introduction

One good reason for using a database is to reduce the overall effort involved in writing and maintaining applications involving large files of complex data. However, some claim that manipulating flat files outside a database provides faster performance. The *database versus flat file* debate is likely to continue indefinitely, but in practice it is **not** the case that flat files are **always** faster than databases. Our purpose here is to present computational evidence that supports our conviction that we should continue building database extension products that assist users in accomplishing the actual tasks they need to perform. For certain applications these products compete well in performance with solutions that avoid the use of databases. Two BCS extensions that considerably enhance the performance of various databases (e.g., Oracle, SQL Server, IBM Informix, and PostgreSQL) are:

1. [DBXten](#), a DataBase eXtension that can be used to efficiently and seamlessly store many types of data (e.g., audio, video, scattered data points, time series, multidimensional data – gridded or non-gridded). It can conveniently and quickly extract any such data that falls within some bounding box.
2. [Grid DataBlade](#), for database applications involving storage, manipulation and extraction of 4D georeferenced gridded datasets. It can resample, reproject, or reorganize data across a rich set of coordinate systems. It supports spatial indexes on the extents of grids, but not on their contents.

Although both the Grid DataBlade and DBXten can handle gridded data, the differences in their functionality are considerable: the Grid DataBlade is a “vertical” product for use with large multidimensional grids, while DBXten is a “horizontal” product with a wider application domain but limited domain-specific functionality.

To illustrate their performance when compared to the direct use of flat files, DBXten and the **Grid Extension** (which is the version of our Grid DataBlade technology that we license for use with PostgreSQL) were used to store a 4.6 GB dataset consisting of a global ocean circulation model hindcast with geographical extent of approximately 25degN to 35degS and 20degE to 140degE. This dataset was downloaded from the [Ocean Circulation and Climate Advanced Modelling \(OCCAM\) Project](#) in netCDF format, which is a common format for ocean model datasets. The downloaded OCCAM data, which was in the form of 81 netCDF files, was then loaded into a DBXten-extended database and into a database equipped with the Grid Extension. Extractions from these databases were performed, and the times to load and extract were observed. These times were then compared with those obtained by managing directly the 81 netCDF files¹.

¹ In comparing the Grid Extension and DBXten *database* solutions with netCDF *file* solutions the only factors considered in this report are speed of ingestion/extraction and storage costs. An even more appealing benefit of using the Grid Extension is that its simple SQL interface removes the complexity of manipulating gridded data (e.g., reprojection and grid fusion) to a much greater degree than would be the case if one were to write such applications using the native netCDF libraries. Similarly, DBXten was designed to handle several forms of complex data in addition to gridded data; consequently, the results contained in this report present only a narrow view of DBXten’s applicability and performance.

About the Data

A set of “5-day mean data” gridded datasets was produced using model run “103” with a resolution of 0.25 degrees. The grid variables (or values) are described at

<http://www.noc.ac.uk/JRD/OCCAM/OME/?page=ferretsrc/griddef#deflist>

and from this dataset the following grid variables and their coordinate variables were stored in the database:

Grid Variables:

| Variable | Description | Type |
|-----------------------|-----------------------|-------|
| Potential Temperature | Mean, degrees Celsius | float |
| Salinity | (PSU-35)/1000 | float |

Coordinate Variables:

| Dimension | Minimum | Maximum | Spacing | Samples |
|-----------------|-------------|-------------|---------|---------|
| Longitude (deg) | 19.875 E | 139.875 E | 0.25 | 481 |
| Latitude (deg) | 35.125 S | 24.875 N | 0.25 | 241 |
| Depth (cm) | 266.83 | 636596.1 | various | 66 |
| Time (day) | 2003/Aug/16 | 2004/Sep/20 | 5 | 81 |

The datasets were produced through a single request to the EMODS Dataserver (see <http://www.noc.soton.ac.uk/JRD/OCCAM/EMODS/select.php>) located at the National Oceanographic Centre in Southampton. The datasets were delivered in the form of 81 netCDF files (one per timestep), with the total disk space required being 4.6 GB.

Computing Test Environment

All table loads and extractions were performed on a Pentium D925 computer, with a clock speed of 3 GHz, 2 GB RAM, 2x2 MB level 2 cache, running Fedora Core 7 and the PostgreSQL 8.2.4 database.

Performance and Tiling

One of the limitations of netCDF is that it doesn't support tiling. This implies that data is stored in the same order as it would be in a multidimensional array in memory. For example, in a 2D array, all the data for the first row would come before all the data for the second row, which in turn would come before all the data for the third row, etc. This is ideal when reading an entire variable, but is less than optimal when trying to read a small rectangular region, as is illustrated overleaf in Figure 1.

This figure depicts a 4x10 grid. The dotted box depicts the 4x4 region of interest to be extracted. The order in which data is stored on the disk is (01,02,03,04,05,06,07,08,09,10,11,12,13,14,...,39,40). The thick gray curves show which parts of the grid would need to be skipped over (when seeking) while extracting the data of interest. In this example, 4 disk seeks (gray arrows) are required.

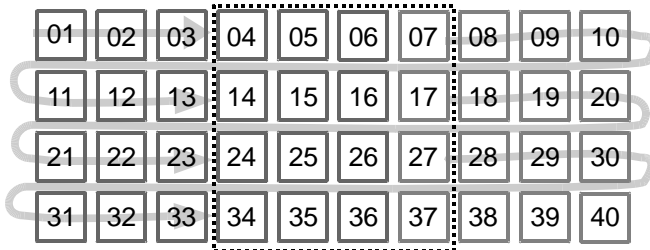


Figure 1: A 4x10 untilted grid of numbers.

In a tiled dataset, the organization of the data on disk is re-arranged to reduce the amount of disk seeking necessary during future extractions. For example, Figure 2 depicts the same 4x10 set of values, but with a 2x3 tiling. The order in which data is stored on the disk is (01,02,03,11,12,13,04,05,06,14,15,16,07,08,09,17,18,19,10,_,_,20,_,_,21,22,23,31,32,33,24,25,26,34,35,36,27,28,29,37,38,39,30,_,_40,_,_)². An attempt to extract data from the region of interest causes 4 tiles to be read in, but only 2 seeks need to be done.

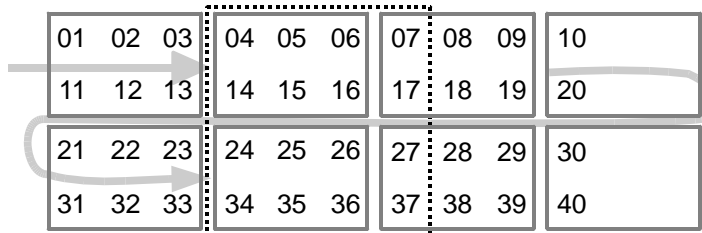


Figure 2: A 2x3 tiling on the 4x10 grid from Figure 1.

The optimal choice of a tile shape balances the gains from reducing the number of disk seeks with the cost of reading in data that is outside the region of interest of a current query. A larger tile can improve the performance of subsequent queries if these queries pertain to the same locality – due to the operating system’s caching of files.

The Grid Extension arranges data into tiles within the blobs it uses to store data. Each tile must be the same size, resulting in wasted space if the tile dimensions don’t divide evenly into the corresponding data dimensions (as shown in Figure 2).

The DBXten technology stores each tile as a separate object and uses GiST³ indexes to locate the tiles that are inside the region of interest. DBXten doesn’t have any requirements about tiles being the same size, so it is much less important that the tile dimensions divide evenly into the data dimensions. Larger tile sizes tend to result in

² The underscores depict portions of the tile for which data is “missing” because the tile extends past the boundaries of the original data.

³ Generalized Search Tree indexes are extensible both in the data types supported and in the queries that can be applied to the data. A particularly appealing feature of GiST is that it allows new data types to be indexed in a manner that supports queries which are natural to the data type.

better compression ratios overall, which decreases storage costs, and can improve apparent I/O speed. Larger tile sizes also decrease the absolute number of tiles, resulting in smaller (better) GiST indexes.

Loading the Data

The data files were loaded using programs which are included with both DBXten and the Grid Extension. In the case of DBXten, each netCDF file was converted into a large number of DBXten tiles, and each tile was written to a row in a single table. For the Grid Extension, each netCDF file was converted into a grid containing a set of tiles, and each grid was written to a single row of a table.

The process of loading tiles into a DBXten database is as follows. A program called ncTileLoader accepts a netCDF file and a simple configuration file. The configuration file maps groups of netCDF variables to the names of tables in the database, specifying the retained precision of each variable and the desired tile dimensions.

The process of loading grids into a Grid Extension database is similar. A program called ncToRawTiles accepts a netCDF file and a configuration file. The program outputs a pair of files (a metadata properties file and a binary data file) for each group of variables described in the configuration file. Each pair of files can then be ingested via a simple SQL statement that makes use of a Grid Extension supplied database function.

The following five types of extractions from 4D oceanographic data sets are typical of user queries, and so they formed the basis of our performance/timing tests:

1. time profiles (e.g., temperature versus day for a specific latitude, longitude, and depth),
2. depth profiles (e.g., temperature versus depth for a specific latitude, longitude and day),
3. latitude-longitude slices (e.g., temperature versus latitude and longitude for a specific depth and day),
4. depth-latitude slices (e.g., temperature versus depth and latitude for a specific time and longitude), and
5. 4D volumes (e.g., temperatures at 4D points for a range of times, depths, latitudes, and longitudes).

In general, no single tiling will be optimal for such a wide range of queries. So, in order to gauge the performance sensitivities of our two database extensions, we experimented with a variety of tilings. Seven different tile sizes/shapes were used with DBXten, and another three with the Grid Extension. Each copy of the dataset was loaded into its own database to simplify administration.

Tilings for DBXten

| Database Name | Time Samples | Depth Samples | Latitude Samples | Longitude Samples |
|---------------|--------------|---------------|------------------|-------------------|
| DBX1 | 1 | 1 | 22 | 37 |
| DBX2 | 1 | 4 | 22 | 37 |
| DBX3 | 1 | 8 | 22 | 37 |
| DBX4 | 1 | 16 | 22 | 37 |
| DBX5 | 1 | 4 | 44 | 37 |
| DBX6 | 1 | 4 | 22 | 74 |
| DBX7 | 1 | 4 | 44 | 74 |

Tilings for the Grid Extension

| Database Name | Time Samples | Depth Samples | Latitude Samples | Longitude Samples |
|---------------|--------------|---------------|------------------|-------------------|
| GE1 | 1 | 1 | 22 | 37 |
| GE4 | 1 | 4 | 22 | 37 |
| GE6 | 1 | 6 | 22 | 37 |

DBXten can exploit the known precision of the data in order to reduce the amount of storage space. So, we made the following (arbitrary) assumptions about precision:

| Column | Precision |
|-------------|-----------|
| Latitude | 0.125 |
| Longitude | 0.125 |
| Depth | 10^{-2} |
| Temperature | 10^{-3} |
| Salinity | 10^{-6} |

Each netCDF file loaded into a DBXten database in 14-16 seconds, and into a Grid Extension database in 10 seconds. The space required for each database is given in the table below; each database table was indexed by a multidimensional index in order to speed retrieval of tiles containing relevant data.

| Database | Number of Table Rows | Table Space Usage (MB) | Index Space Usage (MB) | Total Space Usage (MB) |
|----------|----------------------|------------------------|------------------------|------------------------|
| DBX1 | 764478 | 1689 | 607 | 2303 |
| DBX2 | 196911 | 1575 | 89 | 1670 |
| DBX3 | 104247 | 1508 | 52 | 1566 |
| DBX4 | 57915 | 1538 | 22 | 1566 |
| DBX5 | 107406 | 1642 | 45 | 1693 |
| DBX6 | 106029 | 1631 | 46 | 1684 |
| DBX7 | 57834 | 1614 | 24 | 1644 |
| GE1 | 81 | 3309 | negligible | 3313 |
| GE4 | 81 | 3353 | negligible | 3357 |
| GE6 | 81 | 3621 | negligible | 3625 |

A few explanatory notes are in order here:

- An empty PostgreSQL database, before adding any user tables, requires 4MB of space. This, along with other overhead, accounts for the fact that the sum of the table and index spaces is slightly less than the total space.
- Tree-based indexes (such as GiST) have $O(n \log n)$ space requirements, so reducing the number of tiles that need to be indexed by a factor of k reduces the size of the index by more than a factor of k .
- The amount of index space required with the Grid Extension is negligible here because there are only 81 grids being indexed.
- PostgreSQL appears to apply run length compression to its blobs. This makes it difficult to predict how much space the Grid Extension will require. Without the presence of run length compression, we would expect the Grid Extension databases to require approximately 4.6 GB.

The minimum database space used to store the indexed OCCAM data was 1.53 GB, just 33% of the space taken to store the original netCDF files (4.6 GB). As mentioned above, the database space consumption is dependent on both tile sizes and assumptions about precision, so further improvements in space efficiency might be possible.

Extracting Data from the Database

The process of extracting data from DBXten tiles has the following structure, as depicted in Figure 3. The user determines the region (in world coordinates) of the desired data, and submits a query to the database. The query makes use of GiST indexes to locate the tiles likely to contain the data. The tiles are then decompressed, filtered to just the desired region, and formed into an array, if needed.

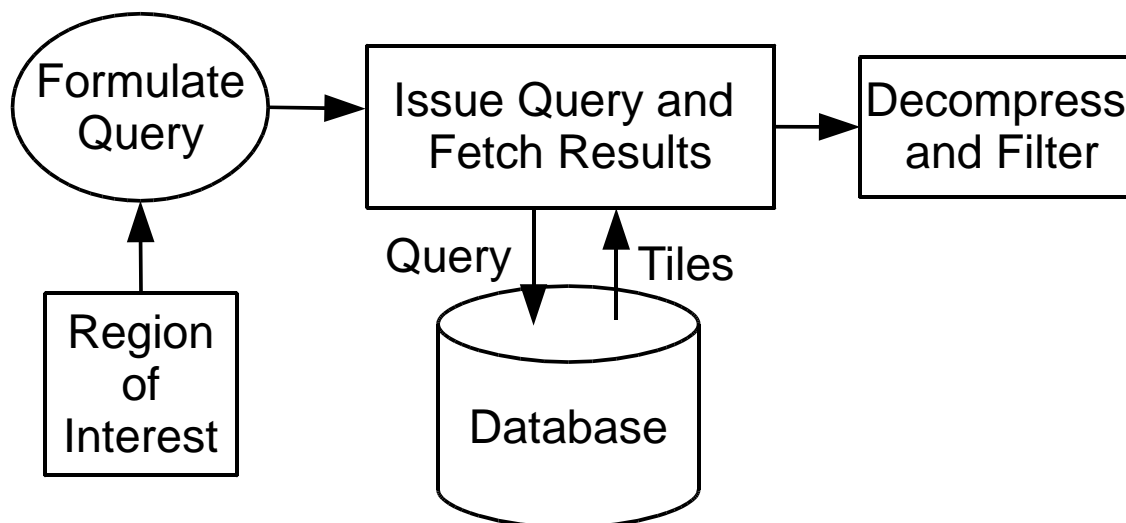


Figure 3: Extracting data from tiles.

The process of extracting data from the Grid Extension objects is almost identical. The only difference is that the data is returned from the database as an object containing one or more scalar fields. Each field has the same organization as a C array.

Extraction Trials

As mentioned above, the types of queries of interest were:

1. time profiles (e.g., temperature versus day for a specific latitude, longitude, and depth),
2. depth profiles (e.g., temperature versus depth for a specific latitude, longitude and day),
3. latitude-longitude slices (e.g., temperature versus latitude and longitude for a specific depth and day),
4. depth-latitude slices (e.g., temperature versus depth and latitude for a specific time and longitude), and
5. 4D volumes (e.g., temperatures at 4D points for a range of times, depths, latitudes, and longitudes).

The tests performed reflected these five types of queries. Each trial consisted of first stopping the PostgreSQL server, clearing the file system's caches, restarting the PostgreSQL server, and then performing the following processing:
 for 1000 times
 extract data from a randomly chosen region.

The average of the elapsed times of the extractions was then computed. To determine the contribution to efficiency of tile configuration, each extraction was performed on the TVARS_DBX table (for the DBXten test program), the TVARS_GRID table (for the Grid Extension test program), and the salinity and potential_temperature variables (in the case of the netCDF program). The five types of queries performed were the following:

| Extraction | Time Samples | Depth Samples | Latitude Samples | Longitude Samples |
|--------------------------|---------------------|----------------------|-------------------------|--------------------------|
| Time Profile | 81 | 1 | 1 | 1 |
| Depth Profile | 1 | 66 | 1 | 1 |
| Latitude-Longitude Slice | 1 | 1 | 50 | 50 |
| | 1 | 1 | 100 | 100 |
| | 1 | 1 | 200 | 200 |
| Depth-Latitude Slice | 1 | 33 | 25 | 1 |
| | 1 | 33 | 50 | 1 |
| | 1 | 66 | 100 | 1 |
| | 1 | 66 | 200 | 1 |
| 4D Volume | 10 | 5 | 20 | 40 |
| | 10 | 10 | 40 | 80 |
| | 10 | 20 | 80 | 160 |

Extraction Timings

The tables below present for each test the average time (in seconds) per extraction.

Tests with DBXten and netCDF files

| Nature of Extraction | Extraction Size | DBX1 1x1x 22x37 | DBX2 1x4x 22x37 | DBX3 1x8x 22x37 | DBX4 1x16x 22x37 | DBX5 1x4x 44x37 | DBX6 1x4x 22x74 | DBX7 1x4x 44x74 | netCDF |
|------------------------|-----------------|-----------------------|-----------------------|-----------------------|------------------------|-----------------------|-----------------------|-----------------------|-------------|
| Time | 81x1x1x1 | 1.82 | 1.21 | 1.13 | 1.03 | 1.12 | 1.09 | 1.03 | 1.30 |
| Depth | 1x66x1x1 | 0.41 | 0.14 | 0.09 | 0.07 | 0.13 | 0.13 | 0.13 | 0.72 |
| Latitude- Longitude | 1x1x50x50 | 0.09 | 0.05 | 0.05 | 0.06 | 0.04 | 0.05 | 0.04 | 0.03 |
| | 1x1x100x100 | 0.10 | 0.07 | 0.07 | 0.09 | 0.06 | 0.06 | 0.05 | 0.03 |
| | 1x1x200x200 | 0.12 | 0.09 | 0.11 | 0.17 | 0.08 | 0.09 | 0.08 | 0.04 |
| Depth- Latitude | 1x33x25x1 | 0.32 | 0.12 | 0.10 | 0.07 | 0.10 | 0.11 | 0.10 | 0.37 |
| | 1x33x50x1 | 0.33 | 0.13 | 0.10 | 0.09 | 0.12 | 0.12 | 0.12 | 0.38 |
| | 1x66x100x1 | 0.47 | 0.23 | 0.17 | 0.14 | 0.20 | 0.22 | 0.21 | 0.80 |
| | 1x66x200x1 | 0.55 | 0.30 | 0.24 | 0.20 | 0.26 | 0.31 | 0.28 | 0.68 |
| 4D Volume | 10x5x20x40 | 0.48 | 0.21 | 0.19 | 0.19 | 0.20 | 0.20 | 0.19 | 0.69 |
| | 10x10x40x80 | 0.66 | 0.30 | 0.29 | 0.34 | 0.31 | 0.31 | 0.34 | 1.24 |
| | 10x20x80x160 | 1.58 | 0.87 | 0.93 | 1.12 | 0.98 | 0.97 | 1.11 | 2.49 |
| Column Average | | 0.58 | 0.31 | 0.29 | 0.30 | 0.30 | 0.30 | 0.31 | 0.73 |

Tests with the Grid Extension and netCDF files

| Nature of Extraction | Extraction Size | GE1 1x1x 22x37 | GE4 1x4x 22x37 | GE6 1x6x 22x37 | netCDF |
|------------------------|-----------------|----------------------|----------------------|----------------------|-------------|
| Time | 81x1x1x1 | 2.49 | 2.44 | 2.24 | 1.30 |
| Depth | 1x66x1x1 | 0.60 | 0.25 | 0.17 | 0.72 |
| Latitude- Longitude | 1x1x50x50 | 0.07 | 0.09 | 0.09 | 0.03 |
| | 1x1x100x100 | 0.10 | 0.12 | 0.13 | 0.03 |
| | 1x1x200x200 | 0.18 | 0.21 | 0.23 | 0.04 |
| Depth- Latitude | 1x33x25x1 | 0.59 | 0.24 | 0.16 | 0.37 |
| | 1x33x50x1 | 0.61 | 0.26 | 0.20 | 0.38 |
| | 1x66x100x1 | 0.90 | 0.49 | 0.38 | 0.80 |
| | 1x66x200x1 | 0.78 | 0.68 | 0.59 | 0.68 |
| 4D Volume | 10x5x20x40 | 0.81 | 0.54 | 0.46 | 0.69 |
| | 10x10x40x80 | 1.61 | 1.09 | 0.99 | 1.24 |
| | 10x20x80x160 | 4.75 | 4.01 | 3.75 | 2.49 |
| Column Average | | 1.12 | 0.87 | 0.78 | 0.73 |

Conclusions

In most cases, the DBXten timings were faster than those for netCDF file manipulation, with the Grid Extension timings usually being intermediate. Only in the particular case of latitude-longitude slices, where the structure of the netCDF file matches the type of extraction being performed, was netCDF file data extraction observed to be faster.

DBXten likely had the best performance in the Time profile test because this test caused the Grid Extension and the netCDF files to read the metadata of 81 different grids or netCDF files, which was an overhead not shared by DBXten.

Our overall conclusion is that PostgreSQL, when enhanced with either of two BCS database extension products, can offer performance that is comparable to that obtained from a C program reading directly from the netCDF files. The Appendix provides details of the testing programs used to produce these performance results.

Acknowledgements

BCS is indebted to Robin Stephens of BMT SeaTech in England and Sjamsul Lakau of BMT Port & Logistics in Singapore for their helpful suggestions on designing appropriate tests, and for their constructive comments on a preliminary version of this report.

Appendix: The Performance Testing Programs

The three performance testing programs shared a common code base to generate a set of 4D boxes for the queries, but used different back ends to perform the actual query. This made it easier to ensure that each program was asked to perform the same set of queries.

In the case of both PostgreSQL based programs, for each 4D box, the back end did the following:

- converted the 4D box to an SQL query,
- executed the query, and
- collected the results.

In the case of the netCDF program, for each 4D box, the back end did the following:

- for each time step included by the 4D box
 - determined which netCDF file corresponded to the time step using a simple arithmetic expression,
 - opened the netCDF file,

- got the list of dimensions from the netCDF file,
- read the values of the variables associated with the dimensions (i.e., the coordinate variables),
- using the coordinate variables, determined the portions of the grid variables to read,
- fetched the desired portion of the salinity variable,
- fetched the desired portion of the potential_temperature variable, and
- closed the netCDF file.

Of course, by expending considerably more effort, there would be some optimizations that could be devised to speed up the netCDF file performance. However, these would probably rely on taking advantage of observed characteristics of the data that may not hold for other datasets.