The Conquest Modeling Framework for Geoscientific Data *

Eddie C. Shek[†], Richard R. Muntz
Data Mining Laboratory
Computer Science Department
University of California, Los Angeles, CA 90024
{shek, muntz}@cs.ucla.edu

November 1, 1994

Abstract

Geoscience studies produce data from various observations, experiments, and simulations at an enormous rate. At UCLA, we are developing the *Conquest* parallel scientific query processing system to effectively support the analysis of complex spatio-temporal geoscientific datasets maintained by heterogeneous data sources in different formats.

In this paper, we describe the Conquest data modeling framework which captures some of the important semantics of geoscientific data and common processing paradigms. The central concept behind the *Conquest data model* is that of the *field*, which is the association of *geometric cells* in a *coordinate space* with dependent variable values. Cells with different semantics and structures can model a large variety of traditional and scientific datasets. The properties of cells also influence the choice of data storage, indexing, and query optimization strategies. In addition to providing a flexible data model to serve as the canonical model for a wide variety of scientific and non-scientific data, it is important for a system to be able to efficiently retrieve and operate on scientific data. As a result, we define an algebra for the Conquest data model in which queries and operations against scientific data can be conveniently expressed. In addition, we present an overview of the Conquest architecture, and some interesting query processing issues related to parallelism, extensibility and heterogeneity.

*This research supported by NASA under HPCC grant NAG-2225.

†Also with Information Sciences Laboratory, Hughes Research Laboratories, Malibu, CA 90265.

1 Introduction

Geoscience studies produce data from various observations, experiments and simulations at an enormous rate. For example, NASA EOS expects to produce over 1 Tbyte of raw data and scientific products per day by the year 2000, and a 100-year UCLA AGCM simulation [16] generates approximately 30 Tbytes of output. However, often only a small percentage of the archived data is ever analyzed. Partly this has been due to the unavailability of sufficient storage and bandwidth at reasonable cost, and partly because of the inadequacies of traditional database management systems. In particular, current database systems fail to deliver the necessary combination of expressiveness, ease of use, flexibility, and efficiency to effectively support the analysis of complex spatio-temporal geoscientific datasets managed by autonomous data sources in many different formats.

At UCLA, we are developing a parallel scientific query processing system called *CONQUEST* (CONtent-based QUEry of Space and Time) to tackle some of the problems presented by geoscientific data and applications. In this paper, we present the logical data modeling framework used in Conquest. The Conquest data model captures several unique characteristics of geoscientific datasets, and hence provides the basis for exploiting these characteristics to parallelize and optimize complex scientific queries. In addition, we have also defined an algebra for the Conquest data model in which queries and operations against scientific data can be conveniently expressed. Geoscientific datasets can be characterized as follows:

- **Geoscientific data measurements are often obtained by sampling over a multi-dimensional space.** Geoscientific data are measured over a multi-dimensional coordinate space in which points (or sets of points in general) are associated with variable values. For instance, UCLA AGCM simulation generates as part of its output a regular 3-dimensional (time-latitude-longitude) sea-level pressure array of pressure values. A central idea behind the Conquest data model is that of the *field*, which is a logical association of cells in a *coordinate space* with dependent variable values. Cells can be of various geometric object types of different dimensionality including points, lines, polygons, and volumes. The set of cells in the coordinate space for which there is an explicit stored variable value is called the *cell coverage* of the field. Some aspects of the structure and topology of the coverage are modeled in order to determine an efficient storage structure and data representation.

- **Geoscientific data are spatio-temporal in nature and include diverse data types.** A large variety of spatio-temporal geoscientific data, ranging from multi-dimensional arrays of floating point numbers (e.g., sea-level pressure dataset) to time series of polygons (e.g., cyclone tracks), need to be brought together and processed in a data analysis application. Spatial (points, lines, polylines, rectangles, polygons, etc.), temporal (absolute time, relative time, etc.), and alpha-numeric (integers, floating point numbers, character strings, etc.) data types are included in the base type system of the Conquest data model. Moreover, the model allows complex geoscientific data types to be defined by combining existing data types through the field construct.

- **Geoscientific datasets of extremely large size are common.** Careful organization of data layout is important in our ability to reap the benefit of parallel processing, and provide efficient retrieval. Tiling a logical dataset into reasonable size chunks which are placed on different disks (e.g., data stripping in RAIDs), is a basis of query and I/O parallelization. Very large scientific datasets are often physically decomposed (tiled or "chunked") into smaller fragments. For example, data collected from a satellite-based sensor are sometimes stored in a set of files, each containing parameter values recorded in a few days. If retrieval is limited to data recorded in one day, only one file need to be accessed. On the other hand, file access can be parallelized if data recorded during a long time frame is to be retrieved.

- **Geoscientific queries are often complex and computationally expensive.** As a result of the large size of scientific data and complexity of queries, optimization and parallelization of scientific queries, and the physical organization of scientific data are extremely important inter-dependent problems that have to be studied in order to deliver performance acceptable by scientists. The Conquest data model captures the tiling, extent, regularity, and ordering of geoscientific datasets, so that these properties can be exploited during query optimization.

- **Geoscientific datasets are often incomplete.** Geoscientific data are often sparse and irregular due to the placement and characteristics of sensors (e.g., the location of an in-situ sensor, the footprint of a satellite-based sensor). In addition, noise and limitations imposed by sensors and communication equipment frequently cause imperfections in observation data. The Conquest algebra includes an operator that allows scientists to specify how to compute missing data values with an *interpolation function* during data analysis. An extreme example of interpolation would be to allow geoscientific simulation models, e.g., UCLA AGCM, to be stored as an interpolation function, while only storing infrequent checkpoints in a field as the basis of "interpolation".

- **Application-specific procedures are often applied to geoscientific datasets during data analysis.** The study of geosciences often involves an elaborate process of exploratory data analysis. In addition to providing a suite of basic data manipulation procedures, our system allows users to extend the environment by introducing complex scientific functions and procedures (e.g., optic flow algorithm for cyclone tracking) to perform application-specific data analysis.

- **Content-based indexing to geoscientific data is often necessary for efficient access.** Another implication of the the large scientific dataset size is the need for content-based indexing to provide convenient access to specific parts of the dataset of interest to scientists. The Conquest data model supports "raw" data fields (e.g., raster images), complex geoscientific features (e.g., cyclone tracks are content-based indexes to the sea-level pressure field), as well as relational data in a uniform data model and query processing framework. It raises the possibility of efficient use of combinations of relational data and geoscientific features as sophisticated content-based indexes to raw field data.

The Conquest data model and algebra serve as the foundation on which our work on query optimization and parallelization, interoperability, storage structure, and high performance I/O in geoscientific data analysis applications is based. In the next section, we define the Conquest data model. Section 3 defines the algebra for the data model and illustrates its use through example queries. Section 4 outlines the Conquest parallel scientific query processing system and some query processing issues. We discuss some work in related areas that have influenced the development of our model in section 5. Finally, section 6 concludes the paper with a brief summary of the Conquest data model.

2 The Conquest Data Model

In this section, we present the Conquest data model, a data modeling framework for geoscientific data management used in the Conquest parallel scientific database system being developed at UCLA. A goal of the data model is to provide a rich but conceptually simple data model to encompass the wide variety of scientific observation and simulation data, while capturing significant semantic and structural properties that have implications on the processing of such data. In addition, the data model is flexible enough to model more conventional data like relations.

2.1 Logical Data Modeling - Fields

The most important concept in the Conquest data model is that of a *field*. A field is a logical construct modeling common scientific datasets in which measurement of scientific parameters are recorded for *cells* in a multi-dimensional *coordinate space*. All cells in a field for which variable values are recorded belong to the same geometric object type in the field's coordinate space. On the other hand, values recorded for cells lie in another multi-dimensional *variable space*.

2.1.1 Coordinate and Variable Attributes

Each field has a tuple of *coordinate attributes*, each tagged with a name and belongs to a basic geometric type. Basic geometric data types to which a coordinate attribute can belong include:

- points(or values) in 1-D space dim_1, `POINT_1D`(dim_1),
- points in 2-D space $dim_1 \times dim_2$, `POINT_2D`(dim_1, dim_2),
- lines in 2-D space $dim_1 \times dim_2$, `LINE_2D`(dim_1, dim_2),

- polylines in 2-D space $dim_1 \times dim_2$, `POLYLINE_2D`(dim_1, dim_2),
- rectangles in 2-D space $dim_1 \times dim_2$, `RECTANGLE_2D`(dim_1, dim_2),
- polygons in 2-D space $dim_1 \times dim_2$, `POLYGON_2D`(dim_1, dim_2),
- circles in 2-D space $dim_1 \times dim_2$, `CIRCLE_2D`(dim_1, dim_2),
- lines in 3-D space $dim_1 \times dim_2 \times dim_3$, `LINE_3D`$(dim_1, dim_2, dim_3)$,
- boxes in 3-D space $dim_1 \times dim_2 \times dim_3$, `BOX_3D`$(dim_1, dim_2, dim_3)$,
- spheres in 3-D space $dim_1 \times dim_2 \times dim_3$, `SPHERE_3D`$(dim_1, dim_2, dim_3)$.

A dimension may be an alphanumeric, spatial, or temporal data type. Examples include integer `INT`, floating point numbers `FLOAT`, boolean value `BOOL`, latitude `LAT`, longitude `LON`, altitude `ALT`, absolute time `ABSTIME`, time interval `INTERVAL`, and relative time `RELTIME`.

The types of the coordinate attributes together determine the type of the cells in a field. In addition, they specify the coordinate space of the field in which cells lie in. Note that many different cell types could be associated with the same coordinate space.

A cell in the coordinate space of a field can be associated with a variable value, which is a tuple of *variable attributes*. The variable value recorded for a cell is not limited to simple data types; it can belong to a complex spatio-temporal data type or even be another field. The types of the coordinate and variable attributes define the type of a field.

As an example, consider a census dataset in which year-end population statistics from 1950 to 1989 are recorded for each of the US states at the end of each year (see Figure 1). Population counts for the states at the end of a year are associated with disjoint polygonal cells each representing a state. The coordinate space of the the census dataset is the 3-dimensional spatio-temporal space `ABSTIME` $\times$ `LON` $\times$ `LAT`. The dataset can be modeled as a field in which cells are 2-dimensional (`LON` $\times$ `LAT`) polygons in time. Cells can be modeled by combining a temporal point attribute `time`, and a polygonal attribute `state` in the 2-dimensional space formed by the longitude and latitude dimensions called `lon` and `lat` respectively. The variable population value recorded for each state at each time can be represented as a tuple with a single integer variable attribute called `pop`.

For the $CENSUS$ field,

- the number of coordinate attributes $CENSUS.ncattr = 2$,
- the temporal coordinate attribute $CENSUS.cattr[1]$ has name $CENSUS.cattr[1].name =$ `time` and type $CENSUS.cattr[1].type =$ `ABSTIME`,

Figure 1: Census data

- the polygonal coordinate attribute $CENSUS.cattr[2]$ has name $CENSUS.cattr[2].name =$ `state` and type $CENSUS.cattr[2].type =$ `POLYGON_2D(LON, LAT)`.
- the cell type $CENSUS.ctype$ contains cells that are polygons-in-time in the coordinate space as dictated by the coordinate attribute types. It is the tuple formed by the coordinate attribute types $ctype[1 \ldots ncattr].type$,

$$\begin{aligned} CENSUS.ctype &= \langle\, CENSUS.cattr[1].type,\ CENSUS.cattr[2].type \,\rangle \\ &= \langle\, \texttt{ABSTIME}, \texttt{POLYGON\_2D(LON, LAT)} \,\rangle, \end{aligned}$$

- the number of variable attributes $CENSUS.nvattr = 1$,
- the sole variable attribute $CENSUS.vattr[1]$ has name $CENSUS.vattr[1].name =$ `pop` and type $CENSUS.vattr[1].type =$ `INT`.
- the variable type $CENSUS.vtype$ is the tuple formed by the variable attribute types $vtype[1 \ldots nvattr].type$,

$$CENSUS.vtype \;=\; \langle\, CENSUS.vattr[1].type \,\rangle \;=\; \langle\, \texttt{INT} \,\rangle,$$

- the field type $CENSUS.type$ is denoted as

$$\begin{aligned} CENSUS.type &= [\, CENSUS.ctype\, ; CENSUS.vtype \,] \\ &= [\, \langle\, \texttt{ABSTIME}, \texttt{POLYGON\_2D(LON, LAT)} \,\rangle\, ; \langle\, \texttt{INT} \,\rangle \,]. \end{aligned}$$

2.1.2 Cell Coverage and Mapping

Not all cells in the coordinate space of a field are associated with variable values. Fo a field F, the *cell coverage* (or simply coverage) $F.ccov \subset F.ctype$ is the set of distinct (but not necessarily disjoint) cells in its coordinate space for which variable values are recorded. Given its cell coverage, the field's *cell mapping* $F.cmap : F.ccov \rightarrow F.vtype$ maps each cell in its cell coverage to a variable value. The cell coverage and cell mapping logically defines a field, which can be denoted as

$$F \;=\; \langle\; F.ccov \;;\; F.cmap \;\rangle.$$

As an example, suppose the cell coverage of the $CENSUS$ field is the set of polygons covering the U.S. states at the end of each year from 1950 to 1989. A part of the field's cell mapping $CENSUS.cmap$ is shown in Table 1.

`time`	`state`	`pop`
1980/12/31	$CA : POLYGON\_2D(...)$	??
1980/12/31	$NY : POLYGON\_2D(...)$	??
1980/12/31	$TX : POLYGON\_2D(...)$	??
...	...	...
1981/12/31	$CA : POLYGON\_2D(...)$	??
1981/12/31	$NY : POLYGON\_2D(...)$	??
1981/12/31	$TX : POLYGON\_2D(...)$	??
...	...	...

Table 1: Portion of the census field's cell mapping $CENSUS.cmap$

2.1.3 Cell Records and Neighborhoods

Since each cell in a field's coverage is associated with a dependent variable value, we often need to refer to both a cell and the variable value associated with it in the same context together as a single data item. For a field F, a *cell record* at a cell is defined as the tuple formed by the cell in a field's cell coverage and its associated variable value,

$$\begin{aligned} F.crec(cell) \;=\; \langle\, cell; F.cmap(cell) \,\rangle \in F.crectype \quad & where \quad cell \in F.ccov, \\ & and \quad F.crectype \;=\; \langle\; F.ctype \;;\; F.vtype \;\rangle. \end{aligned}$$

Sometimes we may want to define a predicate between a cell record in a field F and some data item (of type, say, `SOMETYPE`, which is a data item of any type, but often a cell record in another field) as a means to selecting data in the field,

$$\theta : F.crectype \times \texttt{SOMETYPE} \rightarrow \texttt{BOOL}.$$

Given the predicate *theta* defined on cell records of field F, a subfield of F, F_{sub}, forms a θ-*neighborhood* of a *focus* f if and only if all cell records in it satisfy the condition defined by the predicate θ and the focus, i.e.,

$$\forall cell \in F_{sub}.ccov,\ \theta(F.crec(cell), f) = true.$$

For a focus, there can be many θ-neighborhoods in field F. Among them, the maximal θ-neighborhood is the one which coverage contains those of all other neighborhoods.

For example, we can define a predicate on CENSUS cell records *pop_gt* : $CENSUS.crectype \times$ `INT` $\rightarrow$ `BOOL` which returns whether a state's population exceeds a certain threshold. The maximal *pop_gt*-neighborhood of the value 10000000 in the $CENSUS$ field is a subfield of $CENSUS$ which contains all records of states with more than 10 million residents at the end of a certain year.

2.2 Structure and Semantics of a Field

The coordinate and variable attributes, the cell coverage, and cell mapping logically define a field. On the other hand, we attempt to capture and model some structural properties and semantics of a field which distinguish fields from simple modeling constructs like relations. These structural and semantic properties provide important clues towards query optimization and parallelization, efficient indexing, as well as data representation and physical data organization.

In particular, the Conquest data model models

- the extent within the coordinate space in which all cells lie,
- the spatio-temporal relationship between cells in a field's coverage,
- the logical ordering of cells in the coverage,
- the structure of cell coverage, or in other words, the regularity of values in coordinate attributes.

2.2.1 Extent of Cell Coverage

Coordinate attributes specify the dimensions in a field's coordinate space. In addition, the actual extent within the coordinate space in which cells lie is often needed to construct data structures for storing and supporting efficient access to data. For example, we often need to know the range of coordinate values in all dimensions (i.e., the bounding box of all coordinate values) in the coordinate space to construct a hierarchical spatial data structure like the quadtree.

The *cell coverage extent* of a field is the tuple formed by the extents of its coordinate attributes' values in the field. The extent of a coordinate attribute covering n dimensions is specified as a n-dimensional bounding box containing all coordinate attribute values. A bounding box for a n-dimensional geometric object type is specified by a pair of n-dimensional coordinate points, one with the minimum values for all dimensions, and the other the maximum values. For the $CENSUS$ field, cell coverage extent

$$\begin{aligned} CENSUS.cext &= \langle\ CENSUS.cattr[1].cext, CENSUS.cattr[2].cext\ \rangle \\ &= \langle\ box(\langle 1950/12/31\rangle, \langle 1989/12/31\rangle),\ box(\langle ??^\circ, ??^\circ\rangle, \langle ??^\circ, ??^\circ\rangle)\ \rangle. \end{aligned}$$

2.2.2 Spatio-Temporal Relationship of Cells in Coverage

Cells in the coverage of the $CENSUS$ field are disjoint ($CENSUS.disjoint = true$). In addition, they are sparse ($CENSUS.spacefill = false$) since there are points in the coordinate space which do not lie in any cell in the field's coverage. Sometimes we can take advantage of knowledge concerning the disjointedness and sparsity of cells in the coverage to determine if compact representations of data can be utilized.

Consider a field with non-point cell type whose cell coverage is disjoint and space-filling. Instead of explicitly storing the location and extent of each cell in the coverage, we can associate the cell coverage with a grid ($gridtype = \{rectilinear, triangulation, \ldots\}$) in which cells are defined in relationship to the grid points. Since grid points in such a field would appear in the specification several cells if cells were to be recorded independent, defining cells by the connectivity of grid points eliminates this redundancy and presents the possibility of more compact representation and potentially more efficient processing.

2.2.3 Cell Ordering and Access Pattern

Typically, cells in a coordinate space have a natural order in which they are often accessed. Sequenced geoscientific data are extremely common; many geoscientific data analysis applications involve studying the change of some feature or parameter along certain dimensions. For example, population growth can be tracked by tracing the change of population statistics of each state in time. I/O overhead can be greatly reduced if the physical data layout matches the access pattern.

The natural ordering of cells in a field's coverage, dictated by the access pattern, provide an important hint towards determining the field's physical layout (e.g., sort-ordering, tiling, etc.) to minimize I/O overhead and buffering, and/or introduce parallelization for common queries. In particular, for the $CENSUS$ field, this implies that we may want to sort the population data based on time ($CENSUS.order = incrementbytime$)

2.2.4 Cell Coverage Structure

Consider a field with 2 coordinate attributes of 1-D point type. Dependent data values are associated with points in a 2-dimensional grid. Figure 2 shows the different types of grid regularity [30] that are recognized in our data model. Cells (which are 2-D points in this case) in regular and rectilinear grids are aligned with the axes, while distances between cells along an axis are arbitrary for a rectilinear grid. A structured grid is logically a regular grid subjected to a non-linear transformation. Data recorded on a structured (including regular and rectilinear) grid can be implicitly positioned, which often result in large decrease in storage requirement. On the other hand, cells forming an unstructured grid are randomly scattered and cells have to be explicitly positioned.

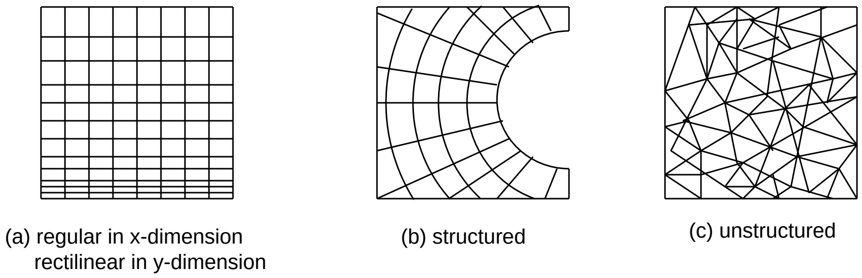


Figure 2: Taxonomy of grid regularity

The Conquest data model captures the regularity of coordinate attribute values in the logical definition of a field. For the i-th coordinate attribute of a field F, the regularity of its values $F.cattr[i].reg$ can be one of the following:

- $regular(start, step, npts)$. Actual values for the ith coordinate attribute appear at regular intervals and can be trivially mapped to and from a regular array index $start + j \times step, j \in \{1, \ldots, npts\}$. In addition to being simple values, coordinate attributes can be multi-dimensional spatial objects, in which case the "shape" of the object would remain the same while its location is displaced by relative position *step*. For instance, the population statistics in the $CENSUS$ field is recorded yearly at year end. As a result, we can model the temporal coordinate attribute of the field as a regular coordinate attribute, $CENSUS.cattr[1].reg = regular(1950/12/31, 1year, 40)$. The range of a regular coordinate attribute determines the extent of its values.

- $rectilinear(func, npts)$. Similar to regular but the regular array index maps to the actual coordinate through a function coordinate attribute values $func(j), j \in \{1, \ldots, npts\}$,

- $structured(func, npts)$. If a field's cell coverage forms a structured grid, we can map coordinates in a regular array $[a_1, \ldots, a_{ncattr}]$ with index ranges $[1 \ldots npts_1] \ldots [1 \ldots npts_{ncattr}]$. to the actual coordinates of the grid $[b_1, \ldots, b_{ncattr}]$ through a non-linear transformation function $b_i = func(a_1, \ldots, a_{ncattr})$.

- *unstructured.* Cells in a grid that is unstructured or with an unknown regularity have to be explicitly positioned. For example, the locations and extents of the U.S. states are unstructured, therefore the second (polygonal) coordinate attribute of the $CENSUS$ field is unstructured, $CENSUS.cattr[2].reg = unstructred$.

2.3 Field Schema

All typing information (e.g., coordinate and variable attribute types), and optional structural and semantic properties (e.g., regularity, cell coverage extent, etc.) of a field constitutes the schema of a field. It can be written as

$$
\begin{array}{ll}
\multicolumn{2}{l}{F(\ cattr[1].name, \ldots, cattr[ncattr].name\ ;\ vattr[1].name, \ldots, vattr[nvattr].name\ ) =} \\
\quad \texttt{field} (& cattr[1].type\ \ [\texttt{EXT} = cattr[1].cext]\ \ [\texttt{REG} = cattr[1].reg],\ \ldots, \\
& cattr[ncattr].type\ \ [\texttt{EXT} = cattr[ncattr].cext]\ \ [\texttt{REG} = cattr[ncattr].reg]\ ; \\
& attr[1].type,\ \ldots,\ vattr[nvattr].type\ ) \\
& [\texttt{DISJOINT}] \\
& [\texttt{SPACEFILL}] \\
& [\texttt{GRIDTYPE} = gridtype] \\
& [\texttt{ORDER} = order]
\end{array}
$$

The range of a regular coordinate attribute determines the extent of its values. As a result, the extent (or bounding box) of a regular coordinate attribute can be left unspecified. In addition, if the regularity of a coordinate attribute is unspecified, we assume that it is unstructured. For example, the schema definition of the $CENSUS$ dataset is written as

```
CENSUS(time,state;pop) =
  field ( ABSTIME  REG=regular(1950/12/31,1 year,10)
        , POLYGON_2D(lon:LON,lat:LAT)  EXT=box(<??',??'>,<??',??'>)
        ; INT ).
  DISJOINT
  ORDER= increment by time
```

2.4 Common Classes of Fields

Complex datasets can be modeled in the Conquest data model by combining different coordinate and variable types each with different properties. Figure 3 shows a simple field hierarchy which shows how common scientific datasets are classified. We will illustrate the different field types with example fields from an atmospheric science application.

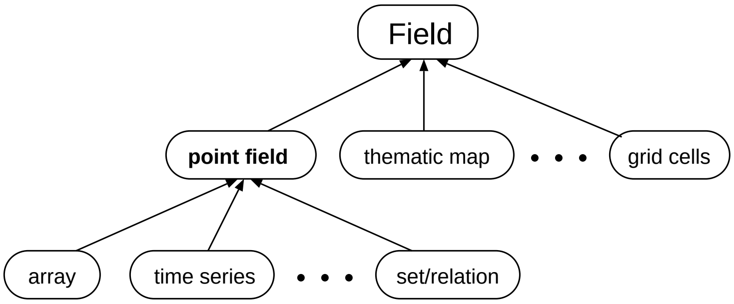


Figure 3: Field Taxonomy

2.4.1 Point Fields

A *disjoint field* is a field in which cell coverage contains a non-overlapping set of cells. A subclass of disjoint fields is the class of *point fields* in which variable values are associated with points (zero-size cells).

Consider a set of fixed observation stations on the earth which record measurements of certain scientific parameters (e.g., sea-level pressure and temperature) recorded at regular intervals (say, every hour on the hour)(see Figure 4). The dataset containing all observation data recorded at the stations is a field with a 3-dimensional coordinate space, in which spatio-temporal points are associated with pressure and temperature values.

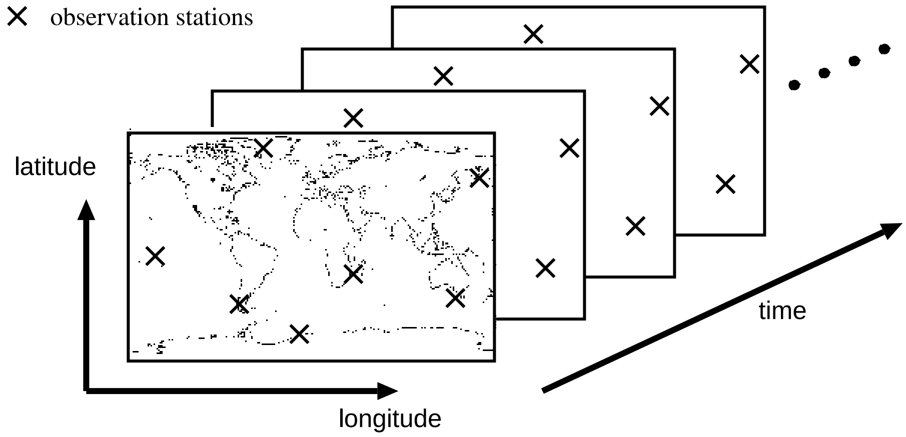


Figure 4: Observation stations

The schema of the observation station dataset can be written as

```
OBS_STATION(time,loc;pressure,temperature)
  = field ( ABSTIME  REG=regular(1980/1/1 00:00,1 hour,87648)
          , POINT_2D(LON,LAT)  EXT=box(<-180',-90'>,<180',90'>)
                               REG=rectilinear(<??',??'>,...)
```

```
        ; FLOAT
        , FLOAT )
    DISJOINT
    ORDER= increment by time.
```

On the other hand, if the observation stations are boats and planes, the location at which pressure and temperature measurements are recorded will be different at different times. In this case, the values of the coordinate attribute `loc` becomes unstructured and it becomes impossible to implicitly position the stations by mapping an integer to the location of the station.

2.4.2 Arrays

Regular multi-dimensional arrays constitute one of the most common types of fields. An array is a point field with a regular and discrete coverage composed of points regularly spaced in all dimensions of the coordinate space.

Consider a 3-dimensional array recording the sea-level pressure at regular grid points at regular intervals in time. For a sea-level pressure field in which data is recorded every day from 1980/1/1 to 1989/12/31 over regular 5° longitude by 4° latitude grids over the surface at the earth, its schema can be written as

```
SLP(time,lon,lat;slp)
  = field ( ABSTIME  REG=regular(1980/1/1,1 day,3652)
        , LON  REG=regular(-180',5',72)
        , LAT  REG=regular(-88',4',45)
        ; FLOAT ).
    DISJOINT
    ORDER= increment by time.
```

The geometric "cells" for which variable sea-level pressure values are recorded are simple 3-dimensional $(time, lon, lat)$ points. The cell coverage of the field is a finite, regular subset of the coordinate space `ABSTIME` $\times$ `LON` $\times$ `LAT`,

$$SLP.ccov = \left\{ \begin{array}{ll} \langle t, lon, lat\rangle \mid & t \in \{1980/1/1, \ldots, 1989/12/31\} \\ & \wedge\ lon \in \{-180^\circ, -175^\circ, \ldots, 175^\circ\} \\ & \wedge\ lat \in \{-88^\circ, -84^\circ, \ldots, 88^\circ\} \end{array} \right\}.$$

A portion of the field's cell mapping (and point mapping) $SLP.cmap$ is shown in Table 2.

A sea-surface temperature field TMP with variable attribute tmp instead of slp can be defined similar to the sea-level pressure field SLP.

t	lon	lat	slp
1/1/93	0°	0°	975
1/1/93	5°	0°	972
1/1/93	10°	0°	977
1/1/93	0°	4°	969
1/1/93	5°	4°	971
1/1/93	10°	4°	972
1/1/93	0°	8°	965
1/1/93	5°	8°	970
1/1/93	10°	8°	969
...	...	...	...

Table 2: Portion of the sea-level pressure field's cell mapping *SLP.cmap*

2.4.3 Time Series

A time series is naturally modeled as a point field with a temporal coordinate space. Consider a type of physical phenomenon which moves over the surface of the earth (e.g., cyclones) as shown in Figure 5. Each trajectory can be viewed as a time series of polygons. For each trajectory the coordinate space is time and the value at a specific time is the spatial object (and possibly other attributes if necessary) representing the phenomenon recorded at that time instant. We can define a field

```
ACYC(time;cyc)
  = field ( ABSTIME  EXT=box(<1980/1/1>,<1989/12/31>)
          ; POLYGON_2D(lon:LON,lat:LAT) )
    DISJOINT
    ORDER= increment by time.
```

to store a cyclone track as a time series of polygons where the polygon recorded at any time covers the extent of the cyclone (according to some definition of extent for a cyclone). The cell coverage *ACYC.ccov* is the set of times in the lifespan of the cyclone at which the extent of the cyclone is recorded. Table 3 shows a portion of the cell mapping of a possible cyclone track field.

If cyclone tracks have unique integer cyclone ids, a set of cyclone tracks can be specified as the nested field

```
CYCS(id;ACYC(time;cyc))
  = field ( INT
```

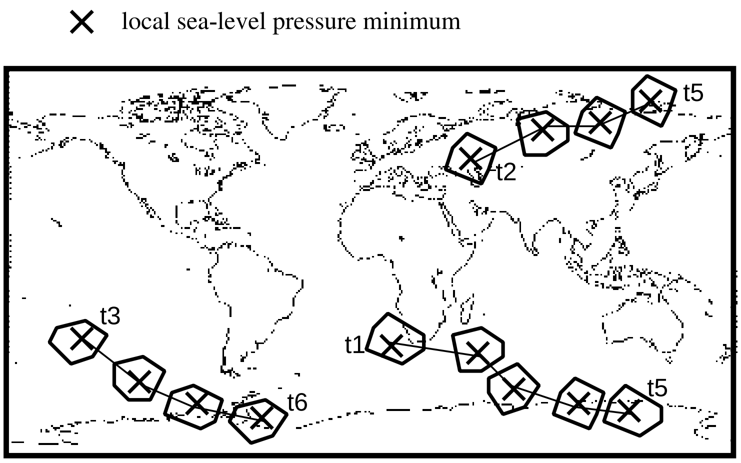


Figure 5: Cyclone tracks

time	cyc
1993/1/1	$POLYGON\_2D\{(0°, 1.1°), (0.5°, 1.0°), (0.6°, 0.7°), (0.3°, 0.9°), (0.1°, 1.0°)\}$
1993/1/2	$POLYGON\_2D\{\ldots\}$
1993/1/3	$POLYGON\_2D\{\ldots\}$
1993/1/4	$POLYGON\_2D\{\ldots\}$
...	...

Table 3: Portion of the cyclone track field's cell mapping $ACYC.cmap$

```
; field ( ABSTIME  EXT=box(<1980/1/1>,<1989/12/31>)
        ; POLYGON_2D(lon:LON,lat:LAT) )
  DISJOINT
  ORDER= increment by time ).
```

The cell coverage $CYCS.ccov$ is the set of cyclone ids, each of which is mapped to a cyclone track field. Table 4 shows a portion of the cell mapping.

2.4.4 Sets and Relations

A set (or a relation without duplicated tuples) can be represented in the Conquest data model as a field with no variable attributes, and in which the types of the unstructured coordinate attributes are equivalent to those of the attributes of the data items. The cell coverage of the field would simply be the set of data items. For example, a set of tuples in relation $R(A_1, \ldots, A_n)$ can simply be modeled as a field with coordinate attributes $A_1, \ldots, A_n$.

id	$ACYC$	
	time	cyc
1	1993/1/1	$POLYGON\_2D\{\ldots\}$
	1993/1/2	$POLYGON\_2D\{\ldots\}$
	1993/1/3	$POLYGON\_2D\{\ldots\}$
	1993/1/4	$POLYGON\_2D\{\ldots\}$
	...	...
2	1993/1/3	$POLYGON\_2D\{\ldots\}$
	...	...
...	...	...

Table 4: Portion of the cyclone tracks' cell mapping $CYCS.cmap$

2.4.5 Thematic Maps

A thematic map[27] is a dataset which contains a spatial and a non-spatial component. Alphanumeric data are associated with geometric objects in the map, specifying their non-spatial attributes and properties.

The $CENSUS$ field, as described earlier in this section, is a classical example of thematic maps. It associates a population value for each state (which is represented as a polygonal region) at each time. For the $CENSUS$ field, the value recorded for a cell is the property of the area subsumed by the cell as a whole. On the other hand, there are cases where the value for a cell is associated with all points in the cell. A land use dataset is a good example of this behavior.

3 Conquest Data Model Algebra

The study of geosciences involves an elaborate process of exploratory data analysis. In this section, we describe the set of basic logical field manipulation operators defined for the Conquest data model. We concentrate on the logical definition of the operators, and only discuss the implementation of an operator when special implementation issues arise. The basic operators are organized into the following classes:

- Relational(-like) operators similar to those in the relational algebra.
- Space conversion operators which derive new fields with different coordinate and variable spaces, as well as new coverages.
- Grouping operators which collect related cell records in a field for further processing.

It is important to note that the algebra presented is not meant to be minimal or complete. Instead, a suite of basic operators is defined to allow scientists to conveniently express their intentions by functionally combining complex scientific data manipulation procedures within the Conquest algebra framework. In addition, specialized application-dependent algorithms are often needed to operate on field data. Therefore we allow users to extend the algebra by introducing new functions, predicates and operators (e.g., optic flow algorithm for cyclone tracking). These new operators and functions will then be available for use by other scientists.

3.1 Relational-like Operators

We begin the specification of the algebra by describing the set-oriented operators that are analogous to the relational operators. The definitions of selection, projection, cartesian product, union, intersection, difference, and join operators are similar to their counterparts in the relational algebra.

3.1.1 Selection σ

The selection operator σ restricts the cell coverage of a field to the set of cells satisfying a selection condition. The selection condition can essentially be any predicate defined on the coordinate and variable attributes of the field, including a neighborhood predicate with an associated focal cell.

Although the schema for the result of a selection is well-defined, the result field often does not inherit the semantic properties (e.g., regularity, range) of the original field. For example, selecting cells in a field based on its variable value (e.g., cells in SLP field at which $slp > 980$) in general returns a field whose cell coverage is unstructured. Selection based on the value of a coordinate attribute may affect the properties of the coordinate attribute, but not those of the other coordinate attributes.

DEF: Given a neighborhood predicate on field F, $\theta : F.crectype \times \texttt{SOMETYPE} \rightarrow \texttt{BOOL}$, and focus $f \in \texttt{SOMETYPE}$, selecting the θ-neighborhood of f from field F is defined as

$$RES \; = \; \sigma_{\theta,f}(F) \; \in \; [F.ctype; F.vtype]$$

where

$$RES.ccov \; = \; \{ \; cell \in F.ccov \; \mid \; \theta(F.crec(cell), f) \; = \; true \; \},$$

$$\forall cell \in RES.ccov, \; RES.cmap(cell) \; = \; F.cmap(cell).$$

3.1.2 Clipping *clip*

Similar to selection which selects all geometric objects in the field that satisfy a certain selection criteria, the *clip* operator selects cells intersecting a particular spatio-temporal "window". A selected cell that is not completely contained in the window is clipped so that only the part of the cell intersecting the window would remain in the cell coverage clipped field. Note that select and clip are equivalent for the point fields as points are zero-size objects and any point intersecting a window has to fall completely inside the window. Figure 6 shows the result of clipping the $CENSUS$ field with a rectangular window.

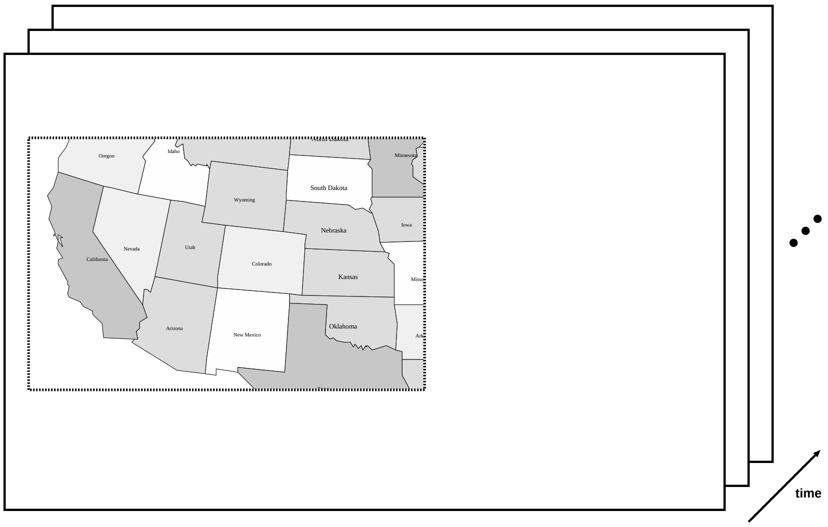


Figure 6: Clipping the census field

DEF: Given a contiguous window $window \subset F.cspace$ in the coordinate space of field F, clipping the field with the window is defined as

$$RES \;=\; clip_{window}(F) \;\in\; [F.ctype; F.vtype]$$

where

$$RES.ccov \;=\; cell = intersect(cell_F, window) \;|\; cell_F \in F.ccov,$$

$$\forall cell_F \in F.ccov,\; RES.cmap(intersect(cell_F, window)) \;=\; F.cmap(cell_F).$$

3.1.3 Cartesian Product $\times$

The coordinate space and variable space of the cartesian product of fields F and G, denoted as $F \times G$, are the cartesian products of those of F and G respectively. The coverages of field F and G

combine to form the coverage of the cartesian product of the fields. Similarly, the cartesian product of their variable spaces becomes the variable space of the fields' cartesian product.

DEF: The cartesian product of fields F and G is defined as

$$RES \;=\; F \times G \;\in\; [F.ctype \times G.ctype; F.vtype \times G.vtype]$$

where

$$RES.ccov \;=\; F.ccov \times G.ccov,$$

$$\forall cell_F \in F.ccov, \forall cell_G \in G.ccov,$$
$$RES.cmap(cell_F \times cell_G) \;=\; F.cmap(cell_F) \times G.cmap(cell_G)\rangle.$$

3.1.4 Union $\cup$, Intersection $\cap$, and Set Difference $-$

Given 2 input fields belonging to the same field space, the union operator collects variable values associated with each cell in the union of their coverages. Because a cell in the common coverage can be associated with different variable values in the input fields, each cell in the union is associated with a multiset containing the variable values associate with it in the input fields.

Similar to union, the intersection and set difference operators collect variable values from cells in input fields. The only difference is that the result coverages are the intersections and set differences of those of the input fields respectively.

DEF: Given fields $F_1, F_2 \in [F.ctype; f.vtype]$, operator $\oplus \in \{\, \cup, \cap, - \,\}$ id defined as

$$RES \;=\; F_1 \oplus F_2 \;\in\; [F.ctype; \{\{F.vtype\}\}]$$

where

$$RES.ccov \;=\; F_1.ccov \oplus F_2.ccov,$$

$$\forall cell \in RES.ccov,\; RES.cmap(cell) \;=\; \{\{\, F_1.cmap(cell), F_2.cmap(cell) \,\}\}.$$

3.1.5 Nesting μ, and Unnesting ν

In the Conquest data model, a nested field is one in which the values associated with cells in the coverage are fields themselves. The nest operator μ moves selected coordinate attributes of a field into the variable space. Each cell in the coverage of the resulting field is associated with a field whose coordinate space is composed of the migrated attributes. Table 5 shows the result of nesting the longitude and latitude attributes of the sea-level pressure field SLP.

time	SLP_FLD		
	lon	lat	slp
1/1/93	0°	0°	975
	5°	0°	972
	10°	0°	977
	...	...	...
1/2/93	0°	0°	970
	5°	0°	982
	10°	0°	975
	...	...	...
...	...	...	...

Table 5: $SLP\_NEST(time; SLP\_FLD(lon, lat; slp)) \; = \; \mu_{SLP\_FLD=lon,lat}(SLP)$

DEF: Given a field $F(a_1, \ldots, a_i; b_1, \ldots, b_j)$, nesting coordinate attributes $c_1, \ldots, c_k$ ($d_1, \ldots, d_l \in \{a_1, \ldots, a_i\} - \{c_1, \ldots, c_k\}$ are the coordinate attributes that are not nested.) into the field's variable space is defined as

$$RES \; = \; \mu_{c_1,\ldots,c_k}(F) \; \in \; [\pi_{d_1,\ldots,d_l}(F.ctype), [\pi_{c_1,\ldots,c_k}(F.ctype), F.vtype]]$$

where

$$RES.ccov \; = \; \pi_{d_1,\ldots,d_l}(F.ccov),$$

$$\forall cell \in RES.ccov, \; RES.cmap(cell) \; = $$
$$\{\langle \; \pi_{c_1,\ldots,c_k}(cell_F), F.cmap(cell_F) \; \rangle \; \mid \; cell_F \in F.ccov, cell = \pi_{d_1,\ldots,d_l}(cell_F)\}.$$

The unnest operator ν has the opposite effect of the nest operator. Consider a nested field $F \in [ctype; [vtype.ctype; vtype.vtype]$. Unnesting the nested field F moves the nested variable space's coordinate attributes $cattr$ into the coordinate space of the field. The result of unnesting F is a field $\nu(F)$ belonging to the field space $[ctype, vtype.ctype; vtype.vtype]$.

To retrieve a variable value $v \in F.vtype.vtype$ nested inside F, we need to go through 2 mappings: first from a cell $c_1 \in F.ccov$ to a field $G = F.cmap(c_1) \in [vtype.ctype; vtype.vtype]$, then from another cell in the coverage of G, $c_2 \in G.ccov$ to the variable value $v = G.cmap(c_2)$.

On the other hand, we can retrieve the same variable value from $\nu(F)$ with only one mapping $v = (\nu(F)).cmap(\langle c_1, c_2\rangle)$. For example, Table 6 shows that repeatedly unnesting the cyclone track field results in the set of all cells in some cyclone track $CYCELL$.

id	time	cyc
0	1/1/93	$POLYGON\_2D(\ldots)$
0	1/2/93	$POLYGON\_2D(\ldots)$
0	1/3/93	$POLYGON\_2D(\ldots)$
...	...	...

Table 6: $CYCELLS(id, time, cyc;) \;=\; \nu(\nu(CYCS))$

DEF: Given a nested field whose variable values are fields, $F \in [F.ctype, [F.vctype, F.vvtype]]$, unnesting the field is defined as

$$RES \;=\; \nu(F) \;\in\; [F.ctype \times F.vctype, F.vvtype]$$

where

$$RES.ccov \;=\; \{\, \langle cell_1, cell_2\rangle \;\mid\; cell_1 \in F.ccov \;\wedge\; cell_2 \in F.cmap(cell_1).ccov \,\},$$

$$\begin{array}{l} \forall cell_1 \in F.ccov,\; cell_2 \in F.cmap(cell_1).ccov, \\ \qquad RES.cmap(\langle cell_1, cell_2\rangle) \;=\; F.cmap(cell_1).cmap(cell_2). \end{array}$$

The nest and unnest operators satisfy the condition that

$$\nu(\mu_{c_1,\ldots,c_k}(F)) = F.$$

By combining the unnest and nest operators, we can freely move attributes between the coordinate and variable spaces. For example, given a field $F(a_1, \ldots, a_i; G(b_1, \ldots, b_j; c_1, \ldots, c_k))$, we can move attributes the attribute b_1 (or any list of attributes in general) into the coordinate space of F by applying the unnest and nest operators to it in succession

$$\mu_{b_2,\ldots,b_j}(\nu(F)).$$

In the Conquest parallel query execution engine, the implementation of the nest and unnest operators is used as one of the mechanisms with which we control the granularity of data communication. In particular, we use cell records as units of data passing between processes. Nesting a field causes "related" cell records in the original field to be grouped in a cell record in the result field. It adds another dimension to our control of the data as we can vary the message size by changing the nesting of the field to be passed.

3.1.6 Natural Join $\bowtie$

Natural join is particularly useful in the Conquest data model. The variable value associated with each cell in the result field's coverage is the tuple formed by the variable values of the corresponding input cells. Given two fields with matching coordinate spaces, it is often useful to increase the dimensionality of the variable space using the natural join, combining the variable values associated with each cell in the input fields' coverages. Table 7 shows the result of naturally joining the sea-level pressure and sea-level temperature fields.

`time`	`lon`	`lat`	`slp`	`tmp`
1/1/93	0°	0°	975	300
1/1/93	5°	0°	972	302
1/1/93	10°	0°	977	303
1/1/93	0°	4°	969	297
1/1/93	5°	4°	971	299
1/1/93	10°	4°	972	299
...	...	...	...	...

Table 7: $SLP \bowtie TMP$

DEF: The natural join of 2 fields F and G is defined as

$$RES \;=\; F \bowtie G \;\in\; [F.ctype \times G.ctype; F.vtype \times G.vtype]$$

where

$$RES.ccov \;=\; F.ccov \bowtie G.ccov,$$

$$\begin{aligned}&\forall cell_F \in F.ccov, cell_G \in G.ccov,\\ &\quad RES.cmap(cell_F \bowtie cell_G) \;=\; \langle F.cmap(cell_F), G.cmap(cell_G)\rangle.\end{aligned}$$

3.1.7 Neighborhood Join $\bowtie_\theta$

In addition to natural join, we allow 2 fields to be joined based on more general join conditions. The join condition is defined as a neighborhood predicate defined on a pair of cell records, one from each input field. Each cell record in the first input field is joined with all cell records in the second input field which lie in its neighborhood. This is similar to θ-join in the relational algebra as the neighborhood predicate can be any predicate defined on the attributes of the input fields. For example, we can join the set of cyclone track cells with sea-level temperature records falling inside the cell extents. Table 8 shows the result of the join.

id	time	cyc	tmp_time	tmp_lon	tmp_lat	tmp
0	1/1/93	$POLYGON\_2D(\ldots)$	1/1/93	0°	0°	300
0	1/1/93	$POLYGON\_2D(\ldots)$	1/1/93	5°	0°	302
0	1/2/93	$POLYGON\_2D(\ldots)$	1/1/93	5°	0°	302
0	1/2/93	$POLYGON\_2D(\ldots)$	1/1/93	0°	0°	300
0	1/2/93	$POLYGON\_2D(\ldots)$	1/1/93	10°	0°	303
0	1/3/93	$POLYGON\_2D(\ldots)$	1/1/93	10°	0°	303
0	1/3/93	$POLYGON\_2D(\ldots)$	1/1/93	5°	0°	302
...	...	...	...	...	...	...

Table 8: $CYCELL\_TMP(id, time, cyc, tmp\_time, tmp\_lon, tmp\_lat; tmp) = CYCELL \bowtie_\theta TMP$ where $\theta \equiv CYCELL.time = TMP.time \wedge within(CYCELL.cyc, TMP.lon, TMP.lat)$

DEF: Given 2 fields F and G, and a neighborhood predicate $\theta : F.crectype \times G.crectype \rightarrow \texttt{BOOL}$, neighborhood join is defined as

$$F \bowtie_\theta G \in [F.ctype \times G.ctype; F.vtype \times G.vtype]$$

where

$$\begin{aligned} RES.ccov \;=\; \cup \{ \langle cell_F, cell_G \rangle \mid\; & cell_F \in F.ccov \wedge cell_G \in G.ccov \\ & \wedge\, \theta(\langle cell_F, F.cmap(cell_F)\rangle, \langle cell_G, G.cmap(cell_G)\rangle) = true \}, \end{aligned}$$

$$\begin{aligned} &\forall cell_F \in F.ccov, cell_G \in G.ccov, \langle cell_F, cell_G\rangle \in RES.ccov, \\ &\quad RES.cmap(\langle cell_F, cell_G\rangle) \;=\; \langle F.cmap(cell_F), G.cmap(cell_G)\rangle. \end{aligned}$$

3.1.8 Coordinate Projection π^C, and Variable Projection π^V

The Conquest data model defines 2 projection operators: coordinate projection π^C and variable projection π^V for projections on the coordinate and variable attributes respectively.

Variable projection behaves similar to relational projection. After variable projection, each cell in the coverage of a field is associated with a projection on the the original variable value.

> **DEF:** Projecting variable attributes $b_1,\ldots,b_l$ from a field F is defined as
>
> $$RES \;=\; \pi^V_{b_1,\ldots,b_l}(F) \;\in\; [F.ctype; \pi^V_{b_1,\ldots,b_l}(F.vtype)]$$
>
> where
>
> $$RES.ccov \;=\; F.ccov,$$
> $$\forall cell \in RES.ccov,\; RES.cmap(cell) \;=\; \pi_{b_1,\ldots,b_l}(F.cmap(cell)).$$

On the other hand, projecting dimensions from the coordinate space is more complicated. Projecting coordinate attributes from a field results in another field with a reduced coordinate space in which each cell is mapped to one or more variable values. The variable values associated with cells in the input field's coverage that project into a cell in the result coverage are collected in a multiset. Table 9 shows the result of projecting out coordinate dimensions related to temperature from the join of cyclone track cells and temperature.

id	time	cyc	TMPS
			tmp
0	1/1/93	$POLYGON\_2D(\ldots)$	300
			302
0	1/2/93	$POLYGON\_2D(\ldots)$	302
			300
			303
0	1/3/93	$POLYGON\_2D(\ldots)$	303
			302
...	...	...	...

Table 9: $CYCELL\_TMPS(id, time, cyc; TMPS\{\{\,tmp\,\}\}) \;=\; \pi^C_{id,time,cyc}(CYCELL\_TMP)$

DEF: Projecting coordinate attributes $a_1,\ldots,a_k$ from a field F is defined as

$$RES \;=\; \pi^{C}_{a_1,\ldots,a_k}(F) \;\in\; [\pi^{C}_{a_1,\ldots,a_k}(F.ctype); \{\{F.vtype\}\}]$$

where

$$RES.ccov \;=\; \pi_{a_1,\ldots,a_k}(F.ccov),$$

$$\forall cell \in RES.ccov,\; RES.cmap(cell) \;=\; \{\{\; F.cmap(\pi_{a_1,\ldots,a_k}(cell_F)) \;\mid\; cell = \pi_{a_1,\ldots,a_k}(cell_F)\;\}\}.$$

Sometimes a user may want to perform projection on a "compound" attribute (e.g., projecting out the year and day from the time attribute in $CYCELL\_TMPS$ in order to collect a multiset of sea-level pressure and temperature values for each month). This can be done by means of space conversion operators. In short, we can apply a conversion function to each time value and map it into a month value.

3.2 Space Conversion

A stumbling block of geoscientific research is the diversity and heterogeneity of data. Consider a 4-dimensional (time-longitude-latitude-altitude) pressure field. First, there are numerous coordinate systems, geographic projections, and georeferencing systems in which data can be recorded. At the same time, we can represent the data with the altitude as a coordinate and pressure as the value, or alteratively have pressure as a coordinate and the altitude at which the pressure is recorded as the value (assuming that pressure monotonically decreases with altitude). As a result of these and other factors, the coordinate and variable spaces of a field often need to be converted in order for the field to be compared to and processed with relevant data. This kind of conversion usually requires interpolation and approximation of field data.

3.2.1 Sampling *sample*

Geoscientific data are often sparse, irregular, and incomplete. Noise and limitations imposed by sensors and communication equipment frequently cause imperfections in observation data. An interpolation function is often necessary to derive missing dependent variable values from recorded ones during data analysis. For example, observation stations are sparse and irregular. Sampling allows a more structured view of the data to be presented to the scientists and hence simplifies queries and applications programs.

The *sample* operator changes the sampling of a field and generates variable values with an *interpolation function* at a user-specified set of cells. For example, a user has to sample the sea-level pressure field if he/she wants to study sea-level pressure at a different resolution from the one assumed by the original field. To simplify computation, scientists often impose a regular grid (for earth surface, the gridding could be equal-size or equal-angle) on a field and perform calculations based on data values associated with the regular grid points through interpolation. Another particularly common use of the sample operator is to change the granularity of the cells in the cell coverage. This can be used to "thin" or "thicken" a rectilinear coordinate space by changing the spacing between grid points in one or more dimensions. Figure 7 shows the effect of changing the granularity of a regular grid.

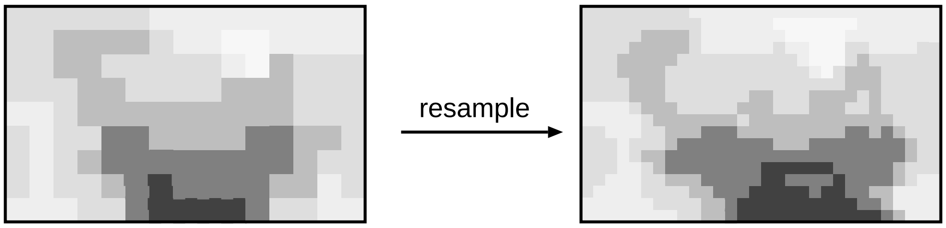


Figure 7: Resampling sea-level pressure field with a finer grid

DEF: Given a field F, a new cell coverage $S \subset F.ctype$ belonging to the field's coverage, and an interpolation function $intrp : F.type \times F.ctype \rightarrow F.ctype$ which evaluates an "interpolated" variable value for a cell given F, sampling is defined as

$$RES \;=\; sample_{S,intrp}(F) \;\in\; [F.ctype; F.vtype]$$

where

$$RES.ccov \;=\; S,$$

$$\forall cell \in RES.ccov,\; RES.cmap(cell) \;=\; intrp(F, cell).$$

3.2.2 Coordinate Space Conversion $convert^C$

A field's coordinate space can be changed by mapping each cell in it into a cell in another space (e.g., convert from sample number in an experiment to actual time, or convert one map projection to another). Alternatively, a user may not want to change the coordinate space but only "move" cells relative to the variable values (e.g. translation or rotation). The coordinate space conversion operator $convert^C$ allows the above mentioned operations on a field to be expressed with a one-to-one coordinate conversion function which changes the value and/or coordinate space of cells in a field.

DEF: Given a one-to-one mapping $\triangleright : F.ctype \rightarrow \texttt{NEWCTYPE}$ from the coordinate space of field F to another coordinate space $\texttt{NEWCTYPE}$, coordinate conversion is defined as

$$RES \ = \ convert^{C}_{\triangleright}(F) \ \in \ [F.ctype; \texttt{NEWCTYPE}]$$

where

$$RES.ccov \ = \ \{ \, \triangleright(cell_F) \ | \ cell_F \in F.ccov \, \},$$

$$\forall cell_F \in F.ccov, \ RES.cmap(\triangleright(cell_F)) \ = \ F.cmap(cell_F).$$

On the other hand, if the coordinate conversion function is many-to-one, multiple coordinates in the input field can be mapped to a single coordinate in the result coverage. In this case, the multiset of variable values associated with the input coordinates is associated with the result coordinate to which the input coordinate is mapped. Note that the coordinate projection operator is a special case of coordinate conversion in which the coordinate function projects out attributes from the coordinate space.

DEF: Given a many-to-one mapping $\triangleright : F.ctype \rightarrow \texttt{NEWCTYPE}$, coordinate conversion is defined as

$$RES \ = \ convert^{C}_{\triangleright}(F) \ \in \ [F.ctype; \{\{\texttt{NEWCTYPE}\}\}]$$

where

$$RES.ccov \ = \ \{ \, \triangleright(cell_F) \ | \ cell_F \in F.ccov \, \},$$

$$\forall cell \in RES.ccov, \ RES.cmap(cell) \ = \\ \{\{ \, F.cmap(cell_F) \ | \ cell_F \in F.ccov \ \wedge \ cell = \triangleright(cell_F) \, \}\}.$$

3.2.3 Variable Space Conversion $convert^V$

The variable space conversion operator $convert^V$ derives new variable values for all cells in the coordinate space of a field by applying a user-defined function (or operator) to each cell record. Each cell in the resulting field is associated with the value returned by the function when applied to the corresponding cell record in the original field. One use of variable space conversion is to perform aggregation on variable values after they are grouped into fields, sets, or multisets by operators that create collections of data values in the variable space of a field. Note that variable projection is a special case of variable conversion in which attributes are projected from the variable space. As an example, Table 10 shows the result of computing the average sea-level temperature over the extent of each cyclone track cell, after temperature values in the neighborhood of each cell of a cyclone track are collected into a multiset in the variable space of field $CYCELL\_TMPS$.

id	time	cyc	avgtmp
0	1/1/93	$POLYGON\_2D(\ldots)$	301
0	1/2/93	$POLYGON\_2D(\ldots)$	301.7
0	1/3/93	$POLYGON\_2D(\ldots)$	302.5
...	...	...	...

Table 10: $CYCELL\_AVGTMP(id, time, cyc; avgtmp) \; = \; convert^V_{avg}(CYCELL\_TMPS)$

DEF: Given a one-to-one mapping $\triangleleft : F.crectype \rightarrow$ `NEWVTYPE` that maps each cell record in field F to some value belonging to type `NEWVTYPE`, variable conversion is defined as

$$RES \; = \; convert^V_{\triangleleft}(F) \; \in \; [F.ctype; \texttt{NEWVTYPE}]$$

where

$$RES.ccov \; = \; F.ccov,$$

$$\forall cell \in RES.ccov, \; RES.cmap(cell) \; = \; \triangleleft(F.crec(cell)).$$

3.3 Cell Record Collection

Data analysis applications often involve the generation of aggregate information on collections of related data from a field. We provide several tuple collection operators for collecting subfields containing related cell records.

3.3.1 Neighborhood Grouping *group*

The grouping operator *group* associates with each cell in a field's coverage a value equal to the subfield composed of all cell records in the field that lie in the neighborhood of that cell. Grouping is usually done in preparation for an operator that will compute a value for each cell based on values in the neighborhood of that cell. For example, Table 11 shows the field produced as a result of grouping all sea-level pressure points within 5° of each sampled coordinate point in a subfield of SLP. A function can that be applied to the field to calculate the average measured sea-level pressure in 5° regions around each point.

time	lon	lat	SLP_FLD			
			time	lon	lat	slp
1/1/93	0°	0°	1/1/93	0°	0°	975
			1/1/93	5°	0°	972
			...	...	...	...
1/1/93	5°	0°	1/1/93	5°	0°	972
			1/1/93	0°	0°	975
			1/1/93	10°	0°	977
			...	...	...	...
1/1/93	10°	0°	1/1/93	10°	0°	977
			1/1/93	5°	0°	972
			...	...	...	...
...	...	...	...	...	...	...

Table 11: $SLP\_GROUP(time, lon, lat; SLP\_FLD(time, lon, lat; slp)) \;=\; group_{within\_5^\circ}(SLP)$

DEF: Given a neighborhood predicate $\theta : F.crectype \times F.crectype \rightarrow \texttt{BOOL}$ defined on a field F with focus being any cell record in F, grouping is defined as

$$RES \;=\; group_\theta(F) \;\in\; [F.ctype; [F.ctype; F.vtype]]$$

where

$$RES.ccov \;=\; F.ccov,$$

$$\forall cell \in RES.ccov,\; RES.cmap(cell) \;=\; \sigma_{\theta,cell}(F).$$

3.3.2 Partition *partition*

In addition to grouping, we allow related cell records in a field to be collected by partitioning the field into a set of subfields containing distinct cells based on a partition function. The cell records in each partition map to the same value in the range of the partition function. Table 12 shows how the sea-level pressure field can be partitioned based the month at which the sea-level pressure value is record. After this partitioning, we can, for example, analyze monthly-variation of sea-level pressure.

month	SLP_FLD			
	time	lon	lat	slp
1	1/1/93	0°	0°	975
	1/1/93	5°	0°	972
	1/1/93	10°	0°	977
	...	...	...	...
	1/2/93	0°	0°	970
	1/2/93	5°	0°	982
	1/2/93	10°	0°	975
	...	...	...	...
2	2/1/93	0°	0°	992
	...	...	...	...
	...	...	...	...

Table 12: $SLP\_PART\{\ SLP\_FLD(time, lon, lat; slp)\ \} \ = \ partition_{month(SLP.time)}(SLP)$

DEF: Given a partition(or hash) function $part\ :\ F.crectype\ \rightarrow$ `NEWCTYPE` for cell records of field F, partition is defined as

$$RES \ = \ partition_{part}(F) \ \in \ [\texttt{NEWCTYPE}; [F.ctype; F.vtype]]$$

where

$$RES.ccov \ = \ \{cell \in \texttt{NEWCTYPE} \ \mid \ cell_F \in F.ccov, cell = part(F.crec(cell_F))\},$$

$$\forall cell \in \texttt{NEWCTYPE}, RES.cmap(cell) \ = \ \sigma_{\lambda crec.\lambda cell.part(crec)=cell,cell}(F).$$

4 The Conquest Parallel Scientific Query Processing System

The Conquest Parallel Query Processing System is an important component of $QUEST$[17], a data analysis environment providing content-based access to massive geoscientific datasets that is under development at UCLA. Figure 8 shows the architecture of QUEST.

In this section, we describe the graphical interface to and the components of the parallel query processing system. We concentrate our discussion on various query processing issues related to parallelism, extensibility, and interoperability.

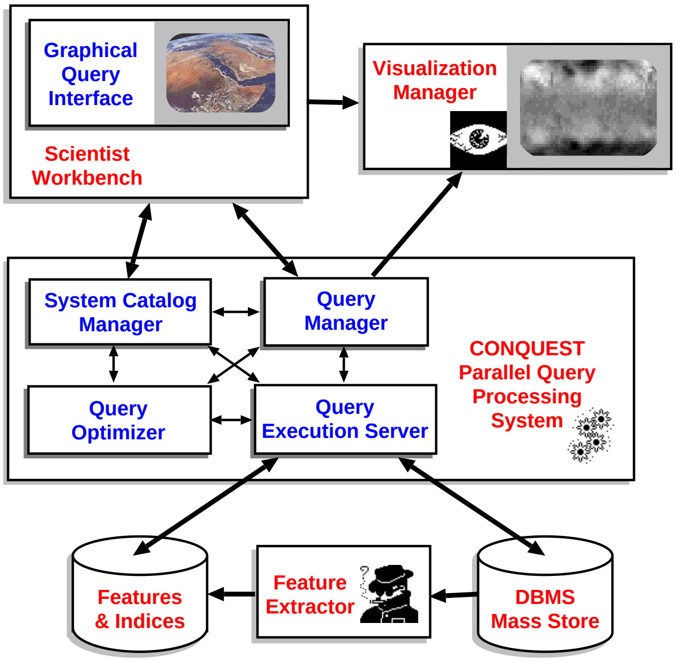


Figure 8: Architecture of QUEST

4.1 Query Manager

A scientist interacts with the Scientist Workbench to specify data analysis queries by locating available datasets of interest and composing queries in the form of dataflow diagrams. A dataflow diagram is converted to an expression in the Conquest algebra, which is then sent to the Conquest Query Manager. The Query Manager, upon receipt of a query, sends the query to the Query Optimizer which generates a parallel Query Execution Plan (QEP). The plan is then forwarded to the Conquest Query Execution Server for evaluation. In the mean time, the Query Manager sets up a connection to the Visualization Manager and initializes it in preparation for the data stream returning from the Query Execution Server as the result of the query.

In addition to our internal storage mechanisms, our environment allows the processing and correlation of scientific data stored in various external autonomous data sources. Possible data sources include external file formats (e.g., HDF [11] for raster data), DBMSs and GISs, distributed CORBA [20] objects encapsulating various data sources, or even “live” scientific data generation processes [18] (e.g., UCLA AGCM simulation).

4.2 Scientist Workbench and Visualization Manager

The QUEST Scientist Workbench provides a graphical interface to a set of scientific data object location, browsing, and manipulation services. As a graphical query interface to Conquest, it provides a visual programming environment, modeled after those supported by popular scientific data analysis and visualization systems (e.g., AVS [1]and IBM Data Explorer [12]), in which scientists interactively manipulate and visualize scientific data. The graphical interface was built using the public-domain Tool Command Language Tcl and X11 Toolkit Tk [21].

In the Scientist Workbench, scientific queries are presented in the form of dataflow diagrams. To construct a dataflow diagram, a user selects operators from an annotated list of available operators (it is possible for a logical operation to have multiple implementations with different parameters and input/output properties) and connects their input and output ports to specify the dataflow. Each query is mapped directly into an algebraic expression in the Conquest algebra and sent to the Conquest Query Manager for evaluation. The use of an intermediate algebraic language for expressing queries provides the basis for multiple query interfaces. In addition to a visual programming environment, we plan on also supporting an SQL3-like query language. in which each query would be parsed and transformed into a Conquest query expression .

After data objects are retrieved from information repositories or through Conquest, they are sent to the QUEST Visualization Manager for visualization. The QUEST Visualization Manager supports static plotting (2D and 3D graphs) of data, analysis of data (e.g., statistical, contours), and animation of datasets. We implemented the visualization manager on top of IDL [23]; IDL was extended with a communication module and support for maintaining the object identifiers of data being manipulated.

4.3 Rule-Based Query Optimizer

We are developing a rule-based query optimizer which takes as input an algebraic expression in the Conquest algebra, and optimize, parallelize, and transform it into a parallel QEP. Both the query expression and the query execution plan are expressions in the Conquest algebra. The difference between a QEP and a logical query is that the QEP is composed of physical operators which are associated with an implementation. This gives knowledgeable users the freedom to “hand-optimize” queries and bypass the Query Optimizer by constructing query execution plans in the GUI and sending them directly to the Query Execution Server. Of course such “queries” have to be parameterized with a great deal of detailed information (e.g., data layout and system configuration) which the optimizer usually handle for the user transparently.

A rule-based query optimizer is characterized by the *execution space* consisting of all semantically equivalent QEPs which correctly evaluate an query, the *cost model* which evaluates the relative “attractiveness” of QEPs, the *rule base* containing rules to transform and generate execution plans, and the *search strategy* which dictates the traversal of the execution space.

The sequential execution space of a query in Conquest is defined by the set of physical operators which implement each logical operator in the query. The *parallel execution space* is also determined by the types of parallelism supported by the support operators and the level of parallelism (e.g., number of processing sites and number of concurrent slave processes that can be executed on the same node) supported by system configuration. The execution space is extended any time a new logical operator or a new implementation of a logical operator is introduced and registered with the System Catalog Server.

During query optimization, transformation and rewrite rules are repeatedly applied to the input query expression. The choice of rule is determined by the search strategy based on some heuristics or cost estimate. An example of a semantic transformation rule specific to scientific query processing is one that replaces a clip operator with a "null" operator that returns no data when the extent of the input field's coverage does not intersect with the clipping window. At the end of optimization, each logical operator is mapped into a physical operator (or a sub-graph of physical operators) which implements the logical definition. Support operators (e.g., sort) are inserted into a QEP to ensure that the physical properties (e.g., sortedness, or for a multi-dimensional array, dimensionality) of the output of a producer operator and the input of a consumer operator match. We are investigating the applicability of different deterministic and randomized search strategies to scientific query parallelization and optimization.

4.4 Parallel Query Execution Server

Scientific datasets have extremely large size and scientific queries often involve the application of computationally expensive operations. System performance for execution of simple queries involving a large amount of data is I/O-bound. However, as operations in a query become more computationally expensive (as in the case of many image processing algorithms), processing speed begins to become a bottleneck. These properties make parallel processing especially attractive for scientific query processing as a means to increase throughput and reduce response time by parallelizing both I/O and computation, rather than just I/O through the use of parallel file systems or RAIDs.

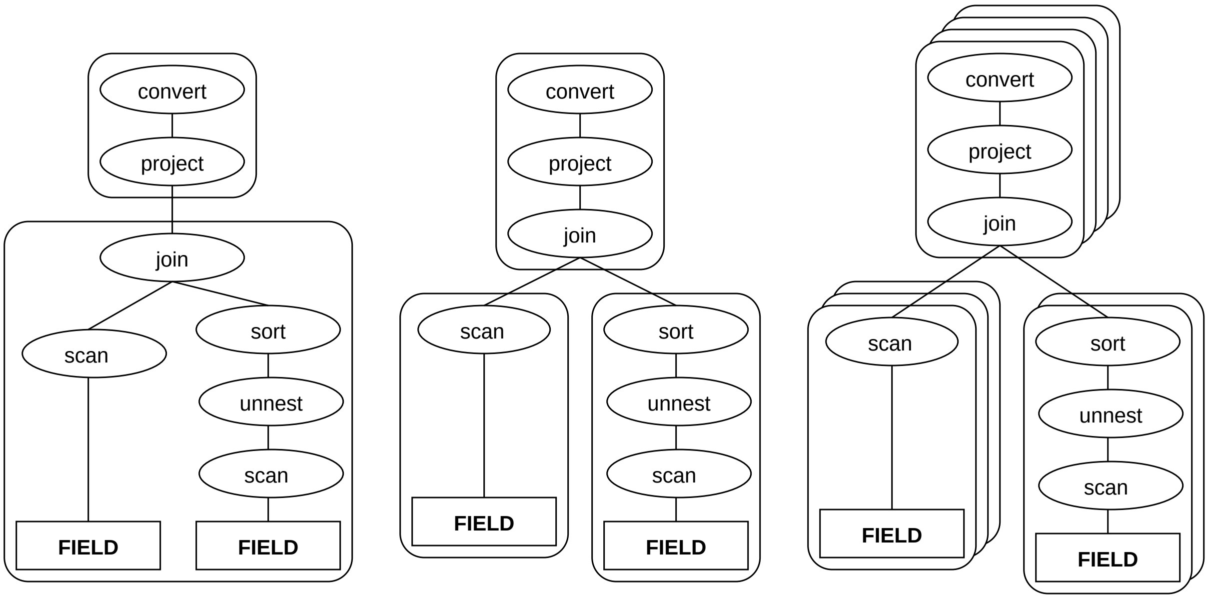
The design and implementation of the Conquest Parallel Query Execution Server are based on the Volcano extensible query execution engine [8]. The Query Executor adopts Volcano's hybrid demand- and data-driven dataflow paradigm, and supports different forms of query parallelization through various support operators. The effectiveness and appropriateness of such a query execution paradigm have been demonstrated for a variety of parallel architectures[9].

The Query Optimizer communicates with the Query Execution Server through the *master scheduler*. When the master scheduler receives a parallel QEP, it propagates the QEP to *slave schedulers* running at processing sites involved in the query execution as specified in the QEP. Upon receipt of a relevant portion of the QEP, a slave scheduler invokes and initializes a number of *slave processes* required for the execution the QEP at that site. Each slave process executes a portion of the QEP. A single thread of control is assigned to all operators executed by a slave process in a demand-driven manner. On the other hand, execution is data-driven across process boundary as the driver operator of a slave process repeatedly calls its producer operator for data and sends data to its consumer process(es) through the communication channel until it fills up.

The following sections describe some of the more interesting features supported by the Conquest Query Execution Server to satisfy the requirements of scientific query processing.

4.4.1 Encapsulation of Parallelism

The Conquest Query Execution Server supports different forms of query parallelization. Figures 9a and 9b show examples of inter-operator parallelism supported by Conquest. Pipeline parallelization is the simplest inter-operator parallelization strategy where processes form a pipeline. Each process obtains data from one producer and feeds it to one consumer after processing. On the other hand, a process involved in bushy parallelization consumes data produced by multiple producer operators executing independently in different processes.



(a) pipeline parallelization (b) bushy parallelization (c) data parallelization

Figure 9: Parallelization of a query execution plan

Intra-operator parallelization provides another dimension of parallelism in addition to inter-operator parallelization. It allows an operator to be executed in several processes, with each copy of the operator handling a (disjoint) subset of the input data. For example, in Figure 9c, a join is executed in 4 concurrent processes. The producer processes partition their output data into 4 subsets and feed each of the subsets to a process containing a copy of the join. The partitioning can be based, for example, on a hash function on the key of a data item.

Conquest encapsulates parallelism through the use of support operators which handle data transfer and distribution between processes, encapsulating interprocess communication from data manipulation operators which implement some logical operator. A notable example is the Volcano *exchange* operator [9]. In addition to the exchange operator, Conquest supports the *split* operator which replicates a data stream produced by an operator to feed multiple consumers, as well as the *partition* operator which partitions scientific data (e.g., raster images) into fragments for processing by multiple consumers in parallel. These logical "no-op"s allow sequential query execution plans to be easily parallelized through the insertion of some parallelization support operators at various points. In addition, they greatly simplify the implementation of user-defined operators as parallelization details are hidden from users.

4.4.2 Dataflow/Stream Processing

A dataflow query processing system achieves pipelined parallelism by connecting operators executed in concurrent processes in series. Data are streamed from the output of the producer to the input of the consumer, allowing computation in the consumer and producer to overlap. The benefit of stream processing parallelism is even more obvious when a scientific query is coupled with a visualization routine which consumes results from the query. In a dataflow processing environment, query results are continually returned without having to wait for the completion of the evaluation of the whole query, reducing the response time and allowing visualization to overlap with query evaluation. Bushy parallelism is similar to pipeline parallelism in that consumers and producers can execute concurrently as data from producers are continually streamed to the consumer. It is particularly useful if data to be correlated or joined are physically partitioned (e.g., stored on different disks or managed by different database servers running on different machines). However, extra care has to be taken to synchronize and coordinate the producers since data may not be produced at the right time or rate.

Pipeline and bushy parallelism naturally supports stream query processing techniques which take advantage of data ordering to deliver excellent performance for many queries. A well-known class of such queries includes various parallel temporal join operations on multi-processor systems [15]. They provide good response time and minimize intermediate storage during parallel query execution. Most geoscientific data have a time component, and scientists are often interested in studying the variation of data in time. As a result, dataflow processing promises to be an attractive evaluation paradigm for scientific queries.

Consider a simple query that evaluates the average sea-level pressure along the path of a cyclone track `ACYC(time:ABSTIME;cyc:POLYGON_2D(lon:LON,lat:LAT))`. The sea-level pressure is stored in field `SLP(time:ABSTIME,lon:LON,lat:LAT;slp:FLOAT)`. The query expression is:

$$ACYC\_AVGSLP(time\ ;\ cyc, avgslp) \ = \ convert^{V}_{avg}(\pi^{C}_{time}(ACYC \Join_{\theta} SLP))$$
$$\text{where} \quad \theta \ \equiv \ (ACYC.time = SLP.time \ \wedge \ within(ACYC.cyc, SLP.lon, SLP.lat))$$

Logically, the cyclone track is joined with sea-level pressure records which lie in the spatial extent of polygonal cyclone cells at each time. Then the sea-level pressure values recorded at points in each cyclone track cell at each time are collected into a field/set by projecting out coordinate attributes from the sea-level pressure field. Finally, the average sea-level pressure over the extent of each cyclone track cell is computed by applying the appropriate aggregate function to the variable values (which is a set of sea-level pressure values) associated with each cell.

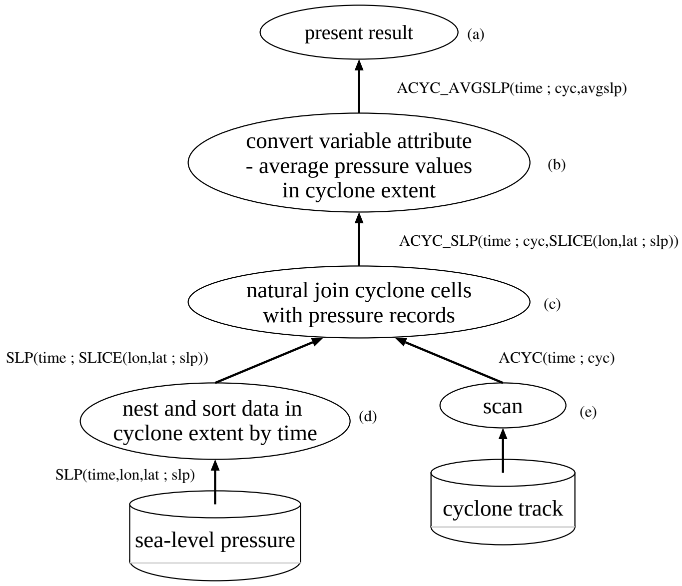


Figure 10: A possible QEP for evaluating average sea-level pressure along cyclone trajectory.

Assuming that cyclone track and sea-level pressure data are stored sorted by time as they are naturally time-ordered, Figure 10 shows a possible query execution plan for the query. The plan is presented in the form of a dataflow diagram which is executed by the dataflow query execution server. In the plan, node (f) scans a spatial-temporal storage structure for cyclone track data. At the same time, operator node (e) scans slices of relevant pressure field data from storage device in time order. The scan is limited to the cell coverage extent of the cyclone track field since any pressure data for points outside of the spatio-temporal window will not be involved in any further processing. Conquest uses cell records as the basic unit of data passing between operators, but for efficiency in message passing, multiple such records are collected into one message. To reduce communication overhead between node (e) and its consumer node (c), scanned pressure data are nested so that each cell record produced at the output of node (e) is a 2-dimensional pressure array recorded at a given time. Such a cell record has a more compact representation than a collection of independent point-value pairs. Node (d) joins retrieved pressure field data with the cyclone track cell recorded at the same time. Since inputs to the operators are sorted by time, I/O overhead due to buffering is minimized. The advantages offered by stream processing is fully exploited as the logical data access pattern matches the physical data organization (both time-ordered). Finally, nodes (b) and (c) evaluate the average sea-level pressure along the cyclone track.

4.4.3 Partitioned Parallelism

For large databases, partitioned parallelism provides opportunities for performance improvement by spreading computation across multiple processors. It is achieved by dividing the input among a set of independent operators, each handling a fragment of the data. Effective data partitioning is an important factor in parallelization of scientific query processing. Various data partitioning schemes (e.g., hash, range, and hybrid-range partitioning) have been developed to fragment relational data among multiple disks to reap the benefits of partitioned parallelism for different access patterns and to reduce the I/O bottleneck by allowing reading and writing of data in parallel.

Scientific datasets often come in the form of large multidimensional arrays. Fragmentation strategies for multidimensional arrays [26] have been proposed to improve access efficiency by tiling arrays into smaller "chunks". Similar techniques can be applied to effectively support partitioned parallelism. Existing techniques partition multidimensional arrays into disjoint fragments. However, scientific applications often involves the correlation of raster elements with neighboring elements (e.g., convolution with a kernel for smoothing, signal processing, shifting, etc.). As a result, partitioning a raster into disjoint fragments and storing each fragment on a different disk result in poor performance for the above-mentioned applications due to the need of copying data elements in the "neighborhood" of of each disjoint fragment to the process responsible for that fragment. Instead, communication overhead can be eliminated by replicating "strips" of neighboring data elements. We are developing new partitioning techniques that partition multidimensional arrays into overlapping fragments, to effectively parallelize queries involving the above mentioned correlation processing. In conjunction with array partitioning techniques, we introduce a raster partition operator which dynamically partitions multidimensional arrays into overlapping (or disjoint if desired) chunks for manipulation by multiple concurrent consumer operators (see Figure 11).

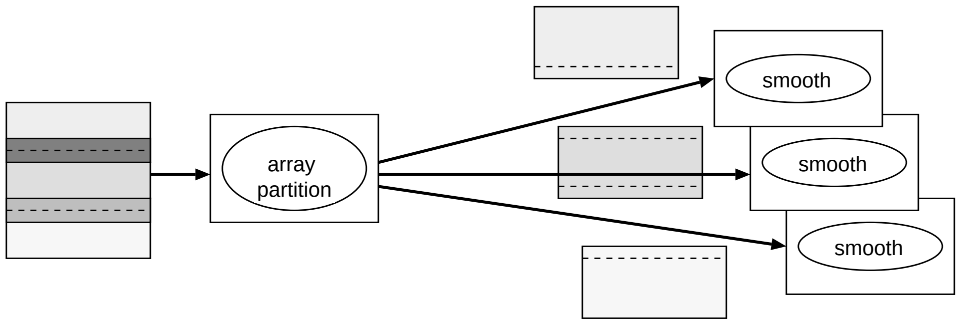


Figure 11: Partitioning a 2D array into overlapping fragments

If data are not physically partitioned or are heavily skewed, it is often beneficial to dynamically redistribute them during query processing, using the split operator to take advantage of intra-operator parallelism and improve load balancing. This is especially useful if the computation involved is expensive and interprocess communication is relatively cheap, which is often the case for scientific data processing in a massively parallel processor environment.

4.4.4 Heterogeneity

Conquest deals effectively with hardware heterogeneity by implementing interprocess communication using version 3.0 of the portable message passing library PVM [7]. PVM enables a network of heterogeneous Unix machines to behave as a single large parallel computer by transparently resolving machine dependence. The Conquest parallel query execution server has been ported to run on the IBM SP1 (a massively parallel processor system composed of a network of interconnected RS6000s) and on Sun workstation farms. Operators in a query execution plan can be executed on processors with different architecture; this is beneficial when the implementations of the operators take advantage of the special benefits offered by the different architectures.

In a distributing computing environment, data structures have to be packaged before transfer to ensure correct transfer between platforms. Conquest provides automatic marshalling and unmarshalling as well as architecture-natural encoding (e.g., XDR) of complex data objects to support transfer of complex scientific data objects between cooperating processes. For a shared-memory environment, data transfer is done through shared memory and marshalling and architecture-natural encoding is skipped to minimize communication overhead.

A large amount of scientific data is already archived in various databases and file formats. It is important for a scientist to be able to access these diverse data sources through a uniform interface. The Conquest Query Execution Server supports access to different databases, file formats, and legacy data sources by allowing users to introduce data source specific access operators (e.g., postgres_scan operator). Currently, we have implemented a generic interface to multi-dimensional raster data in the HDF format [11] and the GRIB format [31]. In addition, we support the use of the extended relational DBMS Postgres [22] as both a storage and an indexing mechanism.

4.4.5 Extensibility

The Query Executor supports an extensible set of physical operators, which include those which implement the Conquest algebra, as defined in previous sections. All physical Conquest operators are encapsulated by a uniform iterator interface (open, next, close, and a few other support functions), allowing new operators to be seamlessly integrated into the system. A user introduces a new operator into Conquest by registering it with the System Catalog Server. An operator is dynamically loaded from a Unix shared library on demand during the initialization of a slave process executing that operator. The fact that parallelism is encapsulated in Conquest through the use of various parallelization support operators contributes to the extensibility of the system as a user only needs to implement a sequential operator, while parallelization issues are handled independently by the support operators. Some of the operators that have been introduced by scientists include general-purpose scientific operations such as Fast Fourier Transform and Butterworth filter, and application-specific feature extraction operators such as cyclone tracking and extraction of blocking features.

In addition to an extensible set of operators, Conquest provides an extensible type system in which user-defined data types and functions can be introduced. We have implemented a large variety of alpha-numeric, spatial, and temporal data types in the built-in basic type system, with which users can construct complex field objects, as defined in the Conquest data model.

4.4.6 Indexing

An important part of the solution to providing efficient scientific data access is indexing. Indexing of scientific data is available at different levels of abstraction. First, a data field, depending on its type, is often naturally associated with a set of index structures and access methods that can be used to support efficient access. A survey of spatial access methods can be found in [25]. In the Conquest Query Execution Server, indexing is encapsulated through various index-scan and index-construct operators.

In addition to traditional index structures and data access methods, content-based indexes provide more sophisticated indexing support which allows scientists to conveniently access specific parts of extremely large scientific datasets. Content-based indexes can be extracted from primitive geoscientific data fields. For example, cyclone tracks, modeled as time-series of polygonal cells on the earth surface, can be used as content-based indexes to a sea-level pressure data field. The Conquest data model is expressive enough to support "raw" data fields, complex geoscientific features, as well as relational data. This allows raw data fields and their content-based indexes to coexist in this system. Supporting scientific data with different roles in a uniform data model and query processing framework introduces the possibility of efficiently utilizing metadata and geoscientific features as sophisticated content-based indexes to raw field data.

4.5 System Catalog Manager

A goal of Conquest is to provide a flexible and extensible environment for scientific query processing. The System Catalog Manager plays a critical role in achieving this goal as it drives the behavior and operation of the different subsystems.

First, the Conquest System Catalog Manager aids the process of data object location and query composition. It maintains an annotated list of logical operators (including operator names, arity, parameter names and types, etc.), a catalog of user-defined data types, and schema information of physical data objects. Users are allowed to browse these metadata through a graphical query interface, and choose for use in their queries and applications.

The Conquest Query Optimizer optimizes a query by mapping the logical algebraic expression into a query execution plan consists of physical operators, and then repeatedly apply QEP transformation rules to generate alternate QEPs. The System Catalog Manager supports extensible rule-based query optimization by maintaining the set of logical-to-physical operator mappings and the set of valid QEP-transformation rules in a rule repository. Additional query optimization specific metadata include the cost model and the physical properties of the inputs/output to each physical operator.

Finally, query parallelization decisions are influenced by the data layout, and system configuration parameters such as the number, architecture, relative performance and load of available resources, as well as the availability of a physical operator implementation on a processor. Metadata about the overall system configuration are also used by the Query Executor to optimize communication. For example, to minimize communication overhead, the Query Executor uses shared memory for data transfer if communicating processes are on the same shared-memory machine.

5 Related Work

The unique data management requirements of scientific applications have been widely recognized [4, 5, 28, 29]. These scientific data modeling and management problems in several areas have been addressed by researchers. In this section, we outline some research that has influenced our approach to geoscientific data management.

5.1 Scientific Data Analysis and Visualization

Some work has been done in developing a data model for scientific visualization [6]. Most notably, the differential geometry notion of *fiber bundle* has been recognized by several researchers as an useful abstraction for an object-oriented scientific visualization data model [2, 3, 10, 14]. In brief, a fiber bundle is a space constructed by attaching a copy of a fiber space to each point in a base space. This "bundling" of dependent variable values to a coordinate space is the central idea behind the Conquest data model. Researchers in scientific visualization modeling concentrates mainly on the mathematical basis of data models. They typically ignore physical data management issues like query optimization, parallelization, and storage layout.

Scientific data analysis and visualization systems (e.g., IDL [23], AVS [1], and Data Explorer [12]), are widely used in the scientific community. IDL provides an array-oriented programming language which allows users to interactively perform a variety of array manipulation operations, making it a useful image processing tool. However, it does not support more complex scientific data structures, hence making it unsuitable for applications involving complex scientific data.

AVS, AVS/Express(an object-based extension to AVS) provide comprehensive suites of mathematical and data analysis functions. Their visual dataflow programming environments allow scientists to interactively manipulate, analyze, and visualize different types of scientific data. AVS/Express' data model supports fields, each of which consists of a mesh (either unstructured, structured, uniform, or rectilinear) of up to 3 dimensions. Each node in the mesh or each cell formed by the node can be associated with a data value. IBM Data Explorer is very similar to AVS and AVS/Express in terms of the capability and programming environment it provides. The grid on which a field is defined is not restricted to 3-dimensional while data can only be associated with grid points or the connections between them.

Field data in these data analysis and visualization systems are closely tied to a grid because implementation is simpler and more easily optimized. However, datasets for which this is not the case (e.g., thematic maps like census data, or cyclone tracks in which cells are not tied to any grid and are somewhat located randomly at different times) are generally not supported. The Conquest data model generalizes the field construct to support a wide variety of data types (including arrays, fields, thematic maps, relations, etc.).

Another important problem faced by these systems is their failure to provide automatic query optimization and parallelization due to the absence of an algebra that unifies the operator suite and exposes opportunities for optimization. Users of such systems have to control every facet of query execution and constantly tune the query (usually in the form of a dataflow diagram) to achieve optimal performance. To overcome this obstacle towards efficient evaluation of scientific queries, we specify an extensible algebra for the Conquest data model supporting many data manipulation operations in which queries are expressed. Algebraic query expressions can be optimized using various algebraic query optimization techniques and rewritten into more efficient query execution plans.

5.2 Geographic Data Interoperability

As a result of the proliferation of geographic applications, standards have been proposed toward interoperability of spatio-temporal data. The Spatial Archive and Interchange Format (SAIF) [24] was designed to be a structurally object-oriented geographic data archival and interchange format. The formulation of a field in the Conquest data model is similar to that of a *coverage* in SAIF, which has a *spatial function* responsible for mapping a position to a value. While SAIF emphasizes the structural composition of spatial objects, it does not address the notions of methods, services, or behavior. It is possible to implement the Conquest field construct as a subclass, say `Conquest_field`, of the SAIF coverage class; while the various special cases of fields can in turn be its subclasses. In this case, operators in the Conquest algebra becomes methods to the `Conquest_field` class, providing behavioral definition to scientific data field objects.

The Open Geodata Interoperability Specification (OGIS) [19] specifies a data model for geographic data, as well as an application programming model aiming to enable communication and interoperability between geographic applications. The Virtual Geodata Model provides an object-oriented model of geographic information which covers at most 3 spatial dimensions and a temporal dimension. Again the formulation of a field in the Conquest data model is similar to that of an OGIS *geographic region* in which non-overlapping, space-filling spatial extents map to dependent data values. Fields in the Conquest data model are more general in the sense that cells in a field can overlap; each cell is associated with some aggregate information for the extent covered by the cell. The OGIS specification is still in a very early stage, but its treatment of behavior interoperability between geographic data stored in different data sources looks promising.

5.3 Object-Oriented and Extended Relational Database Systems

Object-oriented DBMSs and extended relational DBMSs with flexible type systems are gaining in popularity because of their expressiveness; they are much better suited for scientific applications than RDBMSs. Unfortunately, current commercial systems fail to satisfy some of the requirements of scientific data management [13]. For example, OODBMSs have problems with the retrieval of small segments of data within a large data object. More importantly, while *encapsulation* allows the implementation of complex object types to be hidden from the user, it also hides the structure and semantics of the type from the query optimizer. As a result, it is difficult for the optimizer in OODBMSs and extended RDBMSs to exploit the special semantics and structures of scientific data to generate efficient query execution plans. This imposes a severe limitation to the use of such systems for scientific applications considering the size of the data and the complexity of the applications. We try to overcome this limitation by explicitly capturing structural and semantic information about scientific data in the Conquest data model and algebra.

6 Conclusions

In this paper, we have described a flexible data model capturing the unique properties of geoscientific data. The Conquest data model identifies the different roles of coordinates and variable values in geoscientific dataset, hence allow the structure (e.g., regularity, ordering) of coordinates to be be taken advantage of for efficient processing and storage. Moreover, the fact that geoscientific datasets are large in size, and often sparse dictates the need for support of implicitly stored data which are derived by interpolation functions.

Based on the Conquest data model, we have prototyped a parallel dataflow query processor on massively parallel processors as part of the *QUEST* geoscientific data analysis environment that is currently under development at UCLA. Early experiences with the system are promising. The field model is flexible enough to express different geoscientific data types and the query execution engine (with minimal query optimization) delivers performance comparable to hand-tailored programs for many queries.

References

[1] Advanced Visual Systems, Inc., *AVS User's Guide (Release 4)*, May 1992.

[2] D.M. Butler, and M.H. Pendley, "A Visualization Model Based on the Mathematics of Fiber Bundle," *Computers in Physics*, Sep/Oct 1989.

[3] D.M. Butler, and S. Bryson, "Vector-Bundle Classes Form Powerful Tool for Scientific Visualization," *Computers in Physics*, Nov/Dec 1992.

[4] J.C. French, A.K. Jones, and J.L. Pfaltz, "Scientific Data Management," *ACM SIGMOD Record*, Vol. 19, No. 4, Dec 1990.

[5] J. C. French, "Support for Scientific Database Management," *Statistical and Scientific Databases*, Z. Michalewicz (ed.), Ellis Horwood, New York, NY, 1991.

[6] Gallop, J., "Survey of Data Models for Scientific Visualization Systems", *Computer Graphics. Computer Animation, Virtual Reality, Visualization*, 1991.

[7] G.A. Geist, and V.S. Sunderam, "Experiences with Network-Based Concurrent Computing on the PVM System," *Concurrency: Practice and Experience*, Vol. 4, No. 4, Jun 1992.

[8] G. Graefe, "Volcano, an Extensible and Parallel Dataflow Query Processing System," *IEEE Trans. Knowledge and Data Engineering*, Vol. 5, No. 5, Oct 1993.

[9] G. Graefe, and D.L. Davison, "Encapsulation of Parallelism and Architecture-Independence in Extensible Database Query Execution," *IEEE Transactions on Software Engineering*, Vol. 19, No. 8, Aug 1993.

[10] R.B. Haber, B. Lucas, and N. Collins, "A Data Model for Scientific Visualization with Provisions for Regular and Irregular Grids," *Proc. of Visualization '91 Conf.*, San Diego, CA, Oct 1991.

[11] National Center for Supercomputing Applications, *HDF User's Guide, Version 3.2*, 1993.

[12] IBM Corporation, *IBM Visualization Data Explorer: User's Guide (Fifth Edition)*, 1994.

[13] W. Kim, "Object-Oriented Approach to Managing Statistical and Scientific Databases," *Proc. of 5th International Conf. on Statistical and Scientific Database Management*, Charlotte, NC, Apr 1990.

[14] P. Kochevar, Z. Ahmed, J. Shade, and C. Sharp, "A Visualization Architecture for the Sequoia 2000 Project," Sequoia 2000 Technical Report 93/35, Sep 1993.

[15] T.Y.C. Leung, and R.R. Muntz, "Temporal Query Processing and Optimization in Multiprocessor Database Machines", *Proc. of 18th VLDB Conf.*, Vancouver, Canada, Aug 1992.

[16] C.R. Mechoso, C.C. Ma, J.D. Farrara, and J.A. Spahr, "Simulations of Interannual Variability with a Coupled Atmosphere-Ocean General Circulation Model," *Proc. of 5th Conf. on Climate Variations*, American Meteorology Society, Boston, MA, 1991.

[17] E. Mesrobian, R.R. Muntz, E.C. Shek, C.R. Mechoso, J.D. Farrara, and P. Stolorz, "QUEST: Content-based Access to Geophysical Databases", *AAAI Workshop on AI Technologies in Environmental Applications'*, Seattle, WA, Jul-Aug 1994.

[18] E. Mesrobian, R.R. Muntz, E.C. Shek, C.R. Mechoso, J.D. Farrara, J. Spahr, and P. Stolorz, "Real Time Data Mining, Management, and Visualization of GCM Output", *Supercomputing 94 Poster Session*, Washington, DC, Nov 1994.

[19] The OGIS Project Participants, "The Open Geodata Interoperability Specification (Version 1, Draft 1)," Apr 1994.

[20] Object Management Group, "Object Management Architecture Guide (2nd Edition)," Sep 1992.

[21] J. Ousterhout, "Tcl and the Tk Toolkit," Addison Wesley, 1994.

[22] The Postgres Group, "The Postgres 4.2 User Manual," 1994.

[23] Research Systems, Inc., *IDL User's Guide (Version 3.1)*, Jun 1993.

[24] Surveys and Resource Mapping Branch, Ministry of Environment, Lands and Parks, "Spatial Archive and Interchange Format: Formal Definition (Release 3.1)," British Columbia, Canada, Apr 1994.

[25] H. Samet, *Applications of Spatial Data Structures*, Addison-Wesley, Reading, MA, 1990.

[26] S. Sarawagi, M. Stonebraker, "Efficient Organization of Large Multidimensional Arrays," *Proc. of IEEE Conf. on Data Engineering*, Vienna, Austria, Apr 1993.

[27] M. Scholl, and A. Voisard, "Thematic Map Modeling," *Proc. of Symposium on Design and Implementation of Large Spatial Databases*, Santa Barbara, CA, Jul 1989.

[28] A. Shoshani, and H.K.T. Wong, "Statistical and Scientific Database Issues," *IEEE Transactions on Software Engineering*, Vol. 11, No. 10, Oct 1985.

[29] A. Shoshani, "Properties of Statistical and Scientific Databases," *Statistical and Scientific Databases*, Z. Michalewicz (ed.), Ellis Horwood, New York, NY, 1991.

[30] D. Speray, and S. Kennon, "Volume Probes: Interactive Data Exploration on Arbitrary Grids," *Computer Graphics*, Vol. 24, No. 5, Nov 1990.

[31] J.D. Stackpole, "GRIB (Edition 1) - The WMO Format for the Storage of Weather Product Information and the Exchange of Weather Product Messages in Gridded Binary Form", U.S. Department of Commerce, National Oceanic and Atmospheric Administration, Office Note 388, Mar 1993.