Infrastructure for Building Parallel Database Systems for Multi-dimensional Data *

Chialin Chang†, Alan Sussman†, Joel Saltz†+

† Institute for Advanced Computer Studies and Dept. of Computer Science University of Maryland College Park, MD 20742

+ Dept. of Pathology Johns Hopkins Medical Institutions Baltimore, MD 21287

chialin@cs.umd.edu, als@cs.umd.edu, saltz@cs.umd.edu

Abstract

As computational power and storage capacity increase, processing and analyzing large volumes of multi-dimensional datasets play an increasingly important part in many domains of scientific research. Our study of a large set of scientific applications over the past three years indicates that the processing for such datasets is often highly stylized and shares several important characteristics. Usually, both the input dataset as well as the result being computed have underlying multi-dimensional grids. The basic processing step usually consists of transforming individual input items, mapping the transformed items to the output grid and computing output items by aggregating, in some way, all the transformed input items mapped to the corresponding grid point. In this paper, we present the design of *T2*, a customizable parallel database that integrates storage, retrieval and processing of multi-dimensional datasets. T2 provides support for common operations including index generation, data retrieval, memory management, scheduling of processing across a parallel machine and user interaction. It achieves its primary advantage from the ability to seamlessly integrate data retrieval and processing for a wide variety of applications and from the ability to maintain and jointly process multiple datasets with different underlying grids. We also present some preliminary performance results comparing the implementation of a remote-sensing image database using the T2 services with a custom-built integrated implementation.

1 Introduction

As computational power and storage capacity increase, processing and analyzing large volumes of data play an increasingly important part in many domains of scientific research. Typical examples of very large scientific datasets include long running simulations of time-dependent phenomena that periodically generate snapshots of their state (e.g. hydrodynamics and chemical transport simulation for estimating pollution impact on water bodies [5, 7, 23], magnetohydrodynamics simulation

*This research was supported by the National Science Foundation under Grant #ASC 9318183, ARPA under Grant #DABT 63-94-C-0049 (Caltech Subcontract #9503), and the Office of Naval Research under Grant #N6600197C8534. The Maryland IBM SP2 used for the experiments was provided by NSF CISE Institutional Infrastructure Award #CDA9401151 and a grant from IBM. This is a revised and extended version of a paper that appears in ACM SIGMOD Record, March 1998.

of planetary magnetospheres [39], simulation of a flame sweeping through a volume [33], airplane wake simulations [24]), archives of raw and processed remote sensing data (e.g. AVHRR [29], Thematic Mapper [20], MODIS [25]), and archives of medical images (e.g. high resolution confocal light microscopy, CT imaging, MRI, sonography).

These datasets are usually multi-dimensional. The data dimensions can be spatial coordinates, time, or varying experimental conditions such as temperature, velocity or magnetic field. The increasing importance of such datasets has been recognized by several database research groups and several systems have been developed for managing and/or visualizing them [2, 8, 17, 22, 32, 38]. In addition, commercial object-relational database systems provide some support for managing multi-dimensional datasets (e.g., the *SpatialWare DataBlade Module* [36] for the Informix Universal Server and the Oracle 8 *Spatial data cartridge* [31]).

These systems, however, focus on lineage management, retrieval and visualization of multi-dimensional datasets. They provide little or no support for analyzing or processing these datasets – the assumption is that this is too application-specific to warrant common support. As a result, applications that process these datasets are usually decoupled from data storage and management, resulting in inefficiency due to copying and loss of locality. Furthermore, every application developer has to implement complex support for managing and scheduling the processing.

Over the past three years, we have been working with several scientific research groups to understand the processing requirements for such applications [1, 6, 7, 12, 14, 21, 26, 27, 33]. Our study of a large set of applications indicates that the processing for such datasets is often highly stylized and shares several important characteristics. Usually, both the input dataset as well as the result being computed have underlying multi-dimensional grids. The basic processing step usually consists of transforming individual input items, mapping the transformed items to the output grid and computing output items by aggregating, in some way, all the transformed input items mapped to the corresponding grid point. For example, remote-sensing earth images are usually generated by performing atmospheric correction on 10 days worth of raw telemetry data, mapping all the data to a latitude-longitude grid and selecting those measurements that provide the clearest view. As another example, chemical contamination studies simulate circulation patterns in water bodies with an unstructured grid over fine-grain time steps and chemical transport on a different grid over coarse-grain time steps. This is achieved by mapping the fluid velocity information from the circulation grid, possibly averaged over multiple fine-grain time steps, to the chemical transport grid and computing smoothed fluid velocities for the points in the chemical transport grid.

In this paper, we present *T2*, an infrastructure for building parallel database systems that enables integration of storage, retrieval and processing of multi-dimensional datasets. T2 provides support for common operations including index generation, data retrieval, memory management, scheduling of processing across a parallel machine and user interaction. It achieves its primary advantage from the ability to seamlessly integrate data retrieval and processing for a wide variety of applications and from the ability to maintain and jointly process multiple datasets with different underlying grids. Most other systems for multi-dimensional data have focused on uniformly distributed datasets, such as images, maps, and dense multi-dimensional arrays. Many real datasets, however, are non-uniform or unstructured. For example, satellite data consists of a two-dimensional strip that is embedded in a three-dimensional space; water contamination studies use unstructured meshes to selectively simulate regions and so on. T2 can handle both uniform and non-uniform datasets.

T2 has been developed as a set of modular services. Since its structure mirrors that of a wide variety of applications, T2 is easy to customize for different types of processing. To build a version of T2 customized for a particular application, a user has to provide functions to pre-process the input data, map input data to elements in the output data, and aggregate multiple input data items that map to the same output element.

In Section 2 we first present several motivating applications to illustrate their common structure. Section 3 then presents an overview of T2, including its distinguishing features and a running example. Section 4 describes each service in some detail. We discussions how to customize several of the database services for the motivating applications in Section 5. Section 6 provides preliminary performance results for an AVHRR image database system implemented with T2. T2 is a system in evolution. We conclude in Section 7 with a description of the current status of both the T2 design and the implementation of various applications with T2.

2 Motivating examples

Satellite data processing: Earth scientists study the earth by processing remotely-sensed data continuously acquired from satellite-based sensors, since a significant amount of earth science research is devoted to developing correlations between sensor radiometry and various properties of the surface of the earth. A typical analysis [1, 6, 14, 21] processes satellite data for ten days to a year and generates one or more composite images of the area under study. Generating a composite image requires projection of the globe onto a two-dimensional grid; each pixel in the composite image is computed by selecting the "best" sensor value that maps to the associated grid point. A variety of projections are used by earth scientists – the USGS cartographic transformation package supports 24 different projections [40] . An earth scientist specifies the projection that best suits her needs, maps the sensor data using the chosen projection, and generates an image by compositing the projected data. Sensor values are pre-processed to correct the effects of various distortions, such as instrument drift, atmospheric distortion and topographic effects, before they are used.

Virtual Microscope and Analysis of Microscopy Data : The Virtual Microscope [12] is an application we have developed to support the need to interactively view and process digitized data arising from tissue specimens. The Virtual Microscope provides a realistic digital emulation of a high power light microscope. The raw data for such a system can be captured by digitally scanning collections of full microscope slides under high power. The digitized images from a slide are effectively a three-dimensional dataset, since each slide can contain multiple focal planes. At the basic level, the virtual microscope can emulate the usual behavior of a physical microscope including continuously moving the stage and changing magnification and focus. The processing for the Virtual Microscope requires projecting high resolution data onto a grid of suitable resolution (governed by the desired magnification) and appropriately compositing pixels mapping onto a single grid point, to avoid introducing spurious artifacts into the displayed image. Used in this manner, the Virtual Microscope can support completely digital dynamic telepathology [30, 41, 42]. In addition, it enables new modes of behavior that cannot be achieved with a physical microscope, such as simultaneous viewing and manipulation of a single slide by multiple users.

Water contamination studies: Powerful simulation tools are crucial to understand and predict transport and reaction of chemicals in bays and estuaries. Such tools include a hydrodynamics simulator, such as ADCIRC or UTBEST [7, 23], which simulates the flow of water in the domain of interest, and a chemical transport simulator, such as CE-QUAL-ICM [5], which simulates the reactions between chemicals in the bay and transport of these chemicals. For each simulated time step, each simulator generates a grid of data points to represent the current status of the simulated region. For a complete simulation system for bays and estuaries, the hydrodynamics simulator needs to be coupled to the chemical transport simulator, since the latter uses the output of the former to simulate the transport of chemicals within the domain. As the chemical reactions have little effect on the circulation patterns, the fluid velocity data can be generated once and used for many contamination studies. The grids used by the chemical simulator are often different from the grids the hydrodynamic simulator employs, therefore running a chemical transport simulation requires retrieving the appropriate hydrodynamics datasets stored in the database and projecting them into the grid used by the chemical transport simulator. Projection between the grids is performed by a code under development at the University of Texas, called UT-PROJ.

3 Overview

In this section, we provide an overview of T2. We describe its distinguishing features and use a database that generates composite images out of raw satellite data as an example to illustrate how T2 would be used.

There are four distinguishing features of T2. First, it is targeted towards multi-dimensional datasets – the attributes of each dataset form some underlying multi-dimensional attribute spaces (e.g., spatial coordinates, time, temperature, velocity, etc.). T2 can simultaneously manage and process multiple datasets with different attribute spaces and different distributions of data within each attribute space. For example, T2 can manage satellite data at multiple stages in a processing chain ranging from the initial raw data that consists of a two-dimensional strip embedded in a three-dimensional space to ten day composites that are two-dimensional images in a suitable map projection to monthly composites that are 360x180 images with one pixel for each longitude-latitude element. T2 uses multi-dimensional indices (e.g., R^*-trees [3, 16], quad-trees [13]) to manage these datasets. For a given dataset, a separate index is created for every attribute space of interest. For example, the underlying attribute space for AVHRR satellite data has three axes - latitude (in 1/128th of a degree), longitude (1/128th of a degree) and time (in seconds). During processing, this attribute space is mapped to another attribute space, which is a grid in the *Interrupted Goodes Homolosine* map projection [37]. T2 allows users to index this dataset either on the underlying latitude-longitude-time attribute space or on the attribute space jointly defined by the Goodes map projection and time.

Second, T2 leverages commonality in processing requirements to seamlessly integrate data retrieval and processing for a wide variety of applications. Software that integrates data retrieval and data processing can exhibit substantial performance advantages compared to a scenario in which data retrieval and data processing are performed by separate processes, which may be running on different machines. First, integration of data retrieval and computations makes it possible to mask I/O latencies. Second, in many cases, large datasets may have to be copied between the data retrieval program and the data processing program. In our motivating applications, the size of output data for a query are much smaller than the datasets that need to be retrieved for processing the query. T2 integrates data retrieval and processing by providing support for a variety of common operations such as index generation, data retrieval, memory management, scheduling of processing across the parallel machine and user interaction.

Third, T2 can be customized for a wide variety of applications without compromising efficiency. To customize T2, a user has to provide (1) a transformation function to pre-process individual input items; (2) one or more mapping functions to map from the input attribute space to the output attribute space (multiple functions are automatically composed by T2); and (3) an aggregation function to compute an output data item given the set of input data items that map to it.

Fourth, T2 leverages the commonality in the structure of datasets and processing to present a

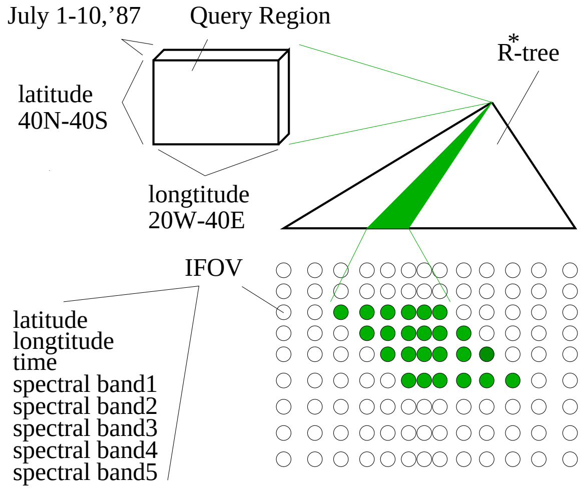


Figure 1: Example of a T2 query for an AVHRR dataset. The query region is specified in terms of the attribute space that underlies the AVHRR dataset.

uniform interface. Users specify in a T2 query the dataset(s) of interest, a region of interest within the dataset(s), and the desired format, resolution and destination of the output. In addition, they select the transformation, mapping and aggregation functions to be used. The output of a T2 query is also multi-dimensional. The attribute space for the output is specified as a part of the query (by specifying the desired format and resolution). The region of interest can be specified in terms of any attribute space that the dataset has an index on. For example, a query to retrieve and process AVHRR data could specify its region of interest in terms of either the latitude-longitude-time attribute space that underlies the AVHRR dataset or the attribute space defined by the Goodes map projection and time.

Figures 1 and 2 show how T2 is used to generate the desired output image from processing raw AVHRR data. Each data item in the AVHRR dataset is referred to as an *instantaneous field of view* (IFOV), and consists of eight attributes – three key attributes that specify the spatio-temporal coordinates and five data attributes that contain observations in different parts of the electromagnetic spectrum. IFOVs from multiple orbits are stored in T2, although Figure 1 only shows a strip from a single orbit. The query region is specified in terms of the latitude-longitude-time attribute space, and an R^*-tree indexed over the IFOVs on the same attribute space is used to identify the IFOVs of interest. The output is an image in the Goodes map projection. Each IFOV selected for the query is pre-processed by a transformation function to correct the effects of various distortions – instrument drift, atmospheric distortion and topographic effects. It is then mapped to a pixel in the output image by a mapping function. Since the query region extends over ten days and since observations from consecutive orbits overlap spatially, multiple IFOVs map to an output pixel. The aggregation function for an output pixel selects the "best" corrected IFOV that maps to the output pixel, based on a measure of the clarity of the sensor readings. Figure 2 illustrates these operations.

4 System Architecture

T2 has been developed as a set of modular services, as shown in Figure 3. Some of the functions provided by these services, such as the indexing service, correspond directly to those provided by

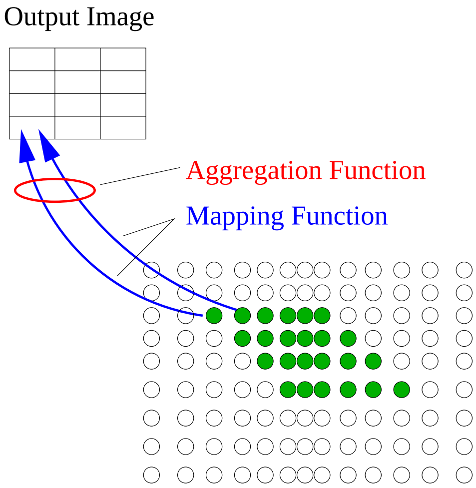


Figure 2: The output of a query into an AVHRR dataset is an image in the Goodes map projection. A transformation function is applied to each IFOV for correction, but is not shown.

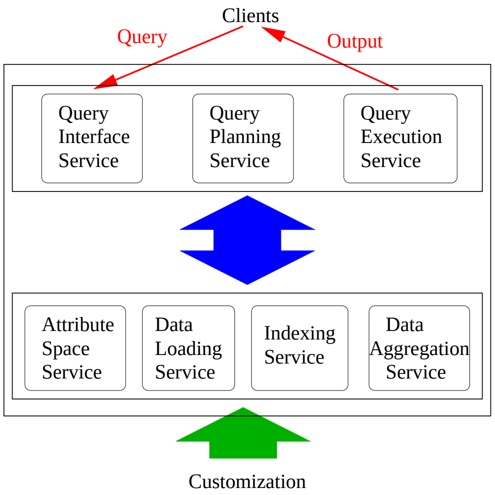


Figure 3: The architecture of T2.

object-relational database systems; other functions are provided to support the stylized processing required by our target applications. While we expect that many applications will be able to use the services as is, we anticipate that some applications may need to replace or modify some of the services.

4.1 The attribute space service

The attribute space service manages the registration and use of attribute spaces and mapping functions. Mapping functions are used to either map individual points between previously registered attribute spaces or to map points from a registered attribute space to define a new attribute space. In this section, we describe how attribute spaces and mapping functions are specified and maintained.

Multi-dimensional attribute spaces are the central structures in T2. All other structures and operations are specified in terms of these attribute spaces. An attribute space is specified by the number of dimensions and the range of values in each dimension. For user convenience, additional information can be stored with an attribute space. For example, a name and the resolution of the values in a dimension can be specified (e.g. for the latitude-longitude-time attribute space from Section 3, the resolution of the latitude dimension is 1/128-th of a degree).

T2 supports two kinds of attribute spaces: *base* and *derived*. *Base* attribute spaces are explicitly specified by the user and are persistent, so can be identified by names that are visible to users. A *derived* attribute space is specified as a (*base* attribute space, mapping function) pair. Logically, a *derived* space is defined as the space generated by mapping every point in the *base* space using the mapping function.

Mapping functions are specified by the domain and range attribute spaces and an algorithm for the mapping between them. The attribute space service manages the namespace of attribute spaces and mapping functions and allows users to browse the sets of available attribute spaces and mapping functions via the query interface service. Currently, mapping functions are statically linked; we plan to provide dynamic linking in the near future.

4.2 The data loading service

The data loading service manages the process of loading new datasets into T2. To load a new dataset into T2, a user has to specify the format, location and metadata for the dataset and the data loading service takes care of loading the dataset and integrating it into the database.

T2 expects incoming datasets to be partitioned into *chunks*, each chunk consisting of one or more data items. T2 allows users to pick any chunk size (all chunks do not have to be the same size); users should pick chunk sizes that allow for efficient retrieval from disk, because chunks are the unit of disk retrieval in T2. The format of the dataset is specified by: (1) the name of a *base* attribute space that underlies the dataset (we call this the *native* attribute space of the dataset); (2) the size of each chunk in the dataset; (3) the number of chunks in the dataset; (4) an iterator function that iterates over the set of data items in a single chunk; and (5) and an access function that given a data item, returns its coordinates in the underlying attribute space.

The location of a dataset is specified as the names of files that contain the dataset. T2 loads each chunk separately, so there are no constraints on the order of the files or on the order of chunks within each file.

The metadata for the dataset consists of placement information. T2 assumes that a disk farm is attached to the processors and placement information is needed to determine the data layout. There are two components of the placement information, both of which are optional. The first is a list of *minimum bounding rectangles* (*mbr*) for each chunk being loaded. An *mbr* for a chunk is a specification of the extent of the data items in the chunk in the attribute space. If the *mbr* information is not specified, it is automatically computed using the iterator and the access functions. The second is a pair of algorithms – one to decluster the chunks to individual disks and the other to cluster them on individual disks. Each algorithm is specified by name. Algorithms that have not been previously integrated into T2 have to be linked in. As for mapping functions, T2 currently supports static linking; we plan to provide dynamic linking in the near future. By default, T2 uses the *minimax* algorithm [26, 27] for declustering and the Short Spanning Path (SSP) algorithm [11] for clustering. In addition, T2 allows the data layout to be separately computed and provided in a file. This would be useful if the algorithms used to compute the placements were embedded in some other application that could not be structured to fit T2's interface requirements.

Once the data layout is specified, the data loading service computes an efficient schedule for moving the chunks to their destinations and executes the schedule.

4.3 The indexing service

The indexing service creates an index for a given (dataset, attribute space) pair. An attribute space can be used for indexing a dataset if and only if it is either the *native* attribute space of the dataset or the target of a chain of mapping functions that maps the *native* attribute space to the new attribute space. T2 allows users to optionally specify an indexing algorithm; by default it uses a variant of R^*-trees.

An index can be created at any time, although it is expected that most indices will be created as a part of the data loading operation. To create an index, the indexing service uses information about the *mbr* for each chunk in the dataset and about the physical location of each chunk on disk. It obtains this information from the data loading service. For *derived* attribute spaces, the indexing service uses the associated mapping function to first map the *mbr* for each chunk into the *derived* attribute space.[1]

4.4 The data aggregation service

The data aggregation service manages the user-provided functions to be used in aggregation operations, and the data types of the intermediate results used by these functions. It manages the namespace of these functions and data types, and performs type checking both when the functions and data types are registered (as a part of customization) and when they are used in response to a query.

An intermediate data structure, referred to as an *accumulator*, may be used during query processing to hold partial results generated by the aggregation functions. It is associated with a data type that provides functions to iterate through its constituent data items, which are called *accumulator elements*. Functions are provided to access the attributes of the individual elements. As for input and output datasets, an accumulator has an underlying attribute space, and each of its elements is associated with a range of coordinates for each dimension of the attribute space. The accumulator data type provides a navigation function that, given a point in the underlying attribute space, returns references to the accumulator elements (there may be more than one) that contain that point. Mapping functions registered with the attribute space service, discussed in Section 4.1, are used together with the navigation function to associate data items from input datasets with accumulator elements. The coordinates of an input data item are first mapped by a chain of mapping functions from the native attribute space of the input dataset to the attribute space for the accumulator, and the navigation function is then used to locate the matching accumulator elements. In addition, an accumulator data type provides the information required by the query planning service, to be described in more detail shortly. T2 currently provides implementations of these functions for accumulators consisting of regular dense arrays with elements evenly spaced in an attribute space, such as raster images. Users, however, can replace these functions with their own implementations.

The data aggregation service also manages functions that implement various aggregation operations. In particular, it manages two kinds of functions: (1) *transformation functions* that take one data item as input and generate another data item as output; and (2) *aggregation functions* that are used to merge the value of an input data item with matching accumulator elements. Transformation functions are used to pre-process data items before aggregation. Aggregation functions are assumed to be commutative and associative and can be applied to individual data items in parallel and in any order. T2 is able to deal with both *distributive* and *algebraic* aggregation functions as defined by Gray et. al [15].

[1] Recall that a *derived* attribute space is specified as a (*base* attribute space, mapping function) pair.

A T2 aggregation function is associated with an accumulator data type, and actually consists of several functions. These include a *data aggregation function* that takes an input data item and a matching accumulator element and aggregates the value of the input data item into the accumulator element. An *initialization function* is used to properly initialize the individual accumulator elements before any aggregation takes place (e.g., with the identity value for the data aggregation function). A *finalization function* postprocesses the accumulator into the desired final output after all the data aggregation has completed. To allow for efficient parallel execution of the aggregation operations, an aggregation function can optionally provide an *accumulator aggregation function*, which merges the values of a set of accumulator elements with another matching set of accumulator elements (e.g., to merge accumulators located on different processors). Such a function allows more flexibility for the query planning service to generate more efficient query execution plans.

Currently, aggregation functions are statically linked. We plan to provide dynamic linking facilities in the near future. Functions are specified by a (function name, object file name) pair. The query interface service uses namespace information from the data aggregation service to allow the user to find the set of transformation functions and aggregation functions that can be applied to a given dataset.

4.5 The query interface service

The query interface service has two functions. First, it allows clients to find out what datasets are available and what functions and indices are associated with each dataset. Second, it allows clients to formulate and present valid queries.

As a part of the first function, the query interface service allows clients to browse all the namespaces in T2: (1) attribute spaces, (2) datasets, (3) indices, (4) placement algorithms, (5) mapping functions, (6) transformation functions, and (7) aggregation functions. As a part of the second function, it ensures that for each query: (1) the domain of the transformation function selected is the same as that of the input dataset (i.e. the types are the same); (2) the range of the transformation function has the same type as the domain of the aggregation function; and (3) the chain of mapping functions is consistent (that is, all the types and shapes match) and the input attribute space of the first mapping function matches the native attribute space of the dataset selected.

4.6 The query planning service

To be able to efficiently integrate data retrieval and processing on a parallel machine, T2 manages the allocation and scheduling of all resources, including processor, memory, disk bandwidth and network bandwidth. The task of the query planning service is to determine a schedule for the use of these resources to satisfy a query. Given the stylized nature of the computations supported by T2, use of several of these resources is not independent (e.g., it is not possible to use disk bandwidth without having memory to store the data being transferred from disk). In addition, the associative and commutative nature of the aggregation operations must be leveraged to form loosely synchronized schedules – the schedules for individual processors need not proceed in lock-step and only need to synchronize infrequently.

The T2 query planning service creates schedules based on requirements for memory, processor and network bandwidth. The input to the planning service consists of: (1) the list of *chunks* that need to be processed, their locations on disk and the region of the output attribute space that each of them maps to; (2) the dependencies between *chunks* – dependencies may occur when multiple datasets are being processed simultaneously; (3) a description of the accumulator, including its size in the underlying attribute space and the extent of each accumulator element in the attribute space; and (4) the amount of memory available on each processor. The output of the planning service consists of a set of ordered lists of *chunks*, one list per disk in the machine configuration. Each list consists of a sequence of sublists separated by synchronization markers. The operations in each sublist can be performed in any order; all operations in a sublist must be completed before any operation in the subsequent sublist can be initiated. This restriction is enforced to ensure schedulability.

We now briefly describe how these resources are taken into consideration during the planning, assuming a shared-nothing database architecture.

Load balancing: the query planning service considers two strategies for load balancing. The first strategy, referred to as *input partitioning*, requires each processor to generate an independent intermediate result in an accumulator using the data aggregation function, based on the chunks that are stored on its local disks. The accumulators are merged using the accumulator aggregation function to obtain the final output. This yields correct results due to the order-independent nature of the processing. The second strategy, referred to as *output partitioning*, partitions the final output; the data needed to compute the portion of the output assigned to a processor is forwarded to it by all the other processors in the machine configuration. The choice between these approaches is based on several factors, including the distribution of the data in the output attribute space, the placement of the input data chunks needed to answer the query on disk, and the machine characteristics (i.e. the relative costs of computation, interprocessor communication and disk accesses). Input partitioning is only possible if the aggregation function selected by the query provides an accumulator aggregation function. However, this strategy often generates less interprocessor communication than output partitioning, since only the accumulators are communicated among the processors, whereas almost all of the input data set must be communicated for output partitioning. For the applications targeted by T2, accumulators are often much smaller than the input data required to satisfy a query. Output partitioning, on the other hand, has a smaller memory requirement than input partitioning, since the accumulators are effectively partitioned among the memories of all processors. Selecting between the two strategies for a given query requires evaluating the tradeoff between communication costs and memory usage.

Memory: T2 uses memory for three purposes – to hold the data read from disk or received from the network, to hold the accumulators for the aggregation operations and to hold the final output. If enough memory is available for all three purposes, operations for all *chunks* in a sublist are scheduled together. Otherwise, memory is first allocated to hold the buffers needed for incoming input data and the remaining memory is partitioned between the other two uses. The planning service can generate a plan that retrieves each input data chunk request just once and bring into memory on demand the required portion of the matching accumulator. When the system runs out of memory during query processing, accumulator elements with partial results must be written back to disk. The advantage of this approach is that all processing for a data chunk can be performed while the chunk is in memory. Alternatively, the planning service can partition the accumulator into chunks that are small enough to fit entirely in memory, and have all the processors work on each accumulator chunk in turn. This approach computes the final output one chunk at a time, and avoids the disk writes generated by the previous approach. However, input data chunks that intersect with multiple accumulator chunks must be retrieved multiple times. These memory management strategies are similar to the strip-mining and/or blocking operations performed for optimizing cache usage for matrix operations [9, 19, 28].

4.7 The query execution service

The query execution service manages all the resources in the system using the schedule created by the planning service. The primary feature of the T2 query execution service is its ability to seamlessly integrate data retrieval and processing for a wide variety of applications. It achieves this in two ways. First, it creates a *query environment* consisting of the set of functions that capture application-specific aspects of the processing. The query environment includes: (1) the access functions for individual input data items; (2) the iterator to iterate over the input data items in a chunk; (3) the transformation function; (4) the sequence of mapping functions that are to be applied to map each input data item to the corresponding accumulator elements; and (5) the aggregation functions needed to compute the output. In effect, explicitly maintaining this environment allows the query execution service to push the processing operations into the storage manager and allows processing operations to be performed directly on the buffer used to hold data arriving from disk. This avoids one or more levels of copying that would be needed in a layered architecture where the storage manager and the processing belonged to different layers.

Second, this service overlaps the disk operations, network operations and the actual processing as much as possible. It does this by maintaining explicit queues for each kind of operation (data retrieval, message sends and receives, processing) and switches between them as required. Appropriate functions are invoked whenever a chunk arrives, either from the local disks or from the network interface. These functions iterate through the data items in a chunk, apply the transformation function to each data item, map the transformed data items to accumulator elements using the mapping function, and finally aggregate the data items that map to each accumulator element. This approach has the advantage of being able to fully overlap the I/O, communication and processing operations even for applications where the amount of work applied to each retrieved data chunk varies widely.

The query execution service performs two kinds of synchronization. First, it enforces the synchronization indicated by the synchronization markers in the list of chunks to be retrieved from every disk (computed by the planning service). That is, the operations between a pair of markers can be performed in any order; all operations before a marker must be completed before any operation after the marker can be initiated. This restriction is used to avoid deadlocks.

The second type of synchronization attempts to preserve load balance by reordering operations. If a particular processor is unable to keep up with its peers, the other processors reorder their operations to reduce the amount of data that is sent to that processor. This mechanism can be used only between synchronization markers.

The last phase of query execution is to compute the final output dataset from the accumulators by invoking the finalization function specified by the given query. The final output is then sent to the destination specified by the query, which could be files on disks or another (potentially parallel) program.

5 Customization examples: AVHRR database, Virtual Microscope, Bay and Estuary Simulation

In this section, we illustrate customization in more detail with three application examples. First, we show customization for the AVHRR satellite database described in Section 3. This example is loosely based on Titan [6], a prototype data server capable of producing composite images out of raw remotely-sensed data. The second customization example is for the Virtual Microscope [12], a system for serving digitized high-power light microscope data. The final example is a system for storing data produced as part of a complete simulation of bay and estuary hydrodynamics and chemistry. In this example, the T2 database stores the output from a hydrodynamics simulation, which can then be used multiple times to perform various simulations of chemical reactions in the bay/estuary, for example to study pollution or oil spill scenarios.

5.1 AVHRR database

The AVHRR dataset is partitioned into IFOV chunks based on the geometry of the IFOVs and the performance characteristics of the disks used to store the data. On the machine used for Titan, one reasonable partitioning creates chunks of 204x204 IFOVs – the size of each chunk is 187 KB. The format of the chunk is specified using an iterator that understands the multi-spectral nature of the values.

The three-dimensional latitude-longitude-time attribute space that underlies the IFOVs is registered as a *base* attribute space with the attribute space service. An access function is used to extract the coordinate attributes from an IFOV, and the coordinates of the four corner IFOVs are used to compute for each chunk a minimum bounding rectangle in the latitude-longitude-time attribute space. The default T2 declustering and clustering algorithms described in Section 4.2 can be used to assign disk locations for the IFOV chunks. The data loading service records all the relevant information about the AVHRR dataset, and moves the IFOV chunks to their assigned disk locations. A simplified R^*-tree suffices for indexing this dataset, and uses the spatio-temporal bounds of the IFOV chunks as access keys. The spatio-temporal bounds are specified as a region in the latitude-longitude-time attribute space. The R^*-tree shown in Figure 1 actually indexes over the IFOV chunks, not the individual IFOVs.

Since the standard AVHRR data product is presented in the Goodes map projection, a three-dimensional attribute space jointly defined by the Goodes map projection and time is registered with the attribute space service as another *base* attribute space, and a mapping function is defined accordingly to map points from the latitude-longitude-time attribute space to this attribute space. This allows the indexing service to map the *mbr* of each IFOV chunk from the latitude-longitude-time attribute space to the Goodes-time attribute space, and build an index for the AVHRR dataset on the Goodes-time attribute space. With this additional index, a query region then can be specified in terms of the Goodes map projection. A two-dimensional spatial attribute space can be derived from either of the three-dimensional spatio-temporal attribute spaces, with a mapping function that discards the temporal coordinate. This derived spatial attribute space is used for the standard AVHRR data product.

As described in Section 3, the transformation function registered with the data aggregation service performs a sequence of corrections to each IFOV. In addition, it also computes the Normalized Difference Vegetation Index (NDVI) [18] for each IFOV, using corrected values from the first two bands of each IFOV. A registered aggregation function selects the NDVI value with the "best" IFOV among all IFOVs that map to a single output pixel, based on the clarity of the IFOV and the angular position of the satellite when the observation was made.

A typical query specifies an area of interest, usually corresponding to a geo-political area of world, and a temporal bound, which gets translated into a query region in either of the two *base* attribute spaces. The query chooses the AVHRR-correction/NDVI-generation algorithm as the transformation function, and the previously described NDVI aggregation algorithm as the aggregation function. The query also specifies the desired resolution of and where to send the output image (e.g., to disk or to another processing program). The query interface service validates the received query, and the query planning service generates an efficient schedule by taking into account the available machine resources. The query execution service carries out data retrieval and processing according to the generated schedule, and sends the output image to the desired destination.

5.2 Virtual Microscope

The Virtual Microscope data set for a slide is a three-dimensional image consisting of an array of RGB pixel values, with two dimensions (**x** and **y**) representing a focal plane within a slide, and the third dimension (**z**) representing multiple focal planes. The data set for a slide is partitioned into rectangular chunks of pixels, with each chunk containing about 100K pixels on the machine that the high performance implementation of the Virtual Microscope currently runs on. The format of an RGB chunk is specified with a simple iterator that allows access to the pixels in raster order, and also understands the layout of the RGB data.

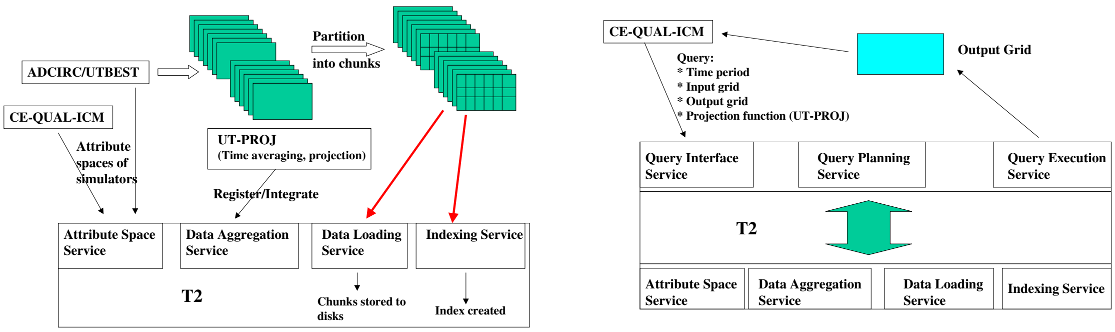
The three-dimensional attribute space underlying the data (effectively a 3D regular grid, with the resolution of the grid determined by the magnification of the digitized microscope image) is registered as a T2 base attribute space. The bounding rectangle of an RGB chunk is computed based on the magnification of the image. The default T2 declustering and clustering algorithms can be used to assign disk locations for the RGB chunks. The data loading service records all the relevant information about the data set for a slide, and builds an index for the data set. The index is a table that is accessed by the (**x,y,z**) location of a data chunk within the slide via a simple computation, and returns the disk location for a given RGB chunk.

The output image for a Virtual Microscope query is a subset of the data for a given slide, at a requested magnification less than or equal to that of the stored data (in powers of two down from the stored image). Therefore three-dimensional attribute spaces for all the lower magnifications are also registered with the attribute space service as derived attribute spaces. A query region can be specified as a subset of the base attribute space for the slide, and the corresponding regions of either the base attribute space or one of the derived spaces for lower magnifications is specified for the output of the query. The mapping function between the base and derived attribute spaces is a straightforward computation to map a region of the higher magnification space to the corresponding pixel in the lower magnification space.

No transformation function for the raw digitized image data is currently required. If some correction to the raw data was desired, it could be specified in the transformation function. A registered aggregation function currently performs subsampling to produce a lower magnification image, although a weighted average of all the pixels that map to the same output pixel could also be used. We are also considering aggregation functions that reconstruct portions of the slide that cut through multiple focal planes, to look at features that cross focal planes. This form of image reconstruction would treat the slide data as a true three-dimensional dataset, allowing better visualization of the data. The cost, on the other hand, is the computation to perform the computational geometry required to determine the data that has to be accessed, and to produce the output image from the stored data.

A typical query specifies a region of the slide in one focal plane at the desired magnification, which gets translated into a query region in the base attribute space. The query chooses the aggregation function (e.g., subsampling or weighted average), and also specifies where to send the output image (e.g., back into T2 on disk, or to a client program). As in the AVHRR database example, the internal T2 services (query interface, query planning and query execution) are then performed to generate the output image that is sent to the desired destination.

The above description of the Virtual Microscope services in T2 assumes that the RGB slide image data is stored in an uncompressed format. We are considering wavelet compression of the



(a) (b)

Figure 4: (a) Putting hydrodynamics simulation output data into T2, and (b) retrieving data from T2 into the chemical transport simulation

image data, both to reduce space requirements in the database, and also to allow for direct retrieval of data at a given magnification without subsampling. Wavelet transformation techniques preserve locality and store data in a multi-resolution format, so it becomes relatively straightforward to reconstruct the desired portion of the slide at the desired magnification with the proper aggregation function. The problem that must be solved before incorporating wavelet compression into the Virtual Microscope is integrating the partitioning of the data into chunks and the compression techniques. This is not just a problem of dealing with variable sized blocks, but also of then being able to index the data so that the parts of the data required to service a query can be found and processed correctly, without having to decompress entire chunks.

5.3 Bay and Estuary Simulation

For this application, the hydrodynamics simulation is used to generate water flow output (e.g., velocities at the grid nodes). These outputs are stored into the T2 database using the data loading service. The chemical transport simulator requests hydrodynamics data from T2 to compute transport of the various chemicals being simulated. T2 performs the data retrieval of the hydrodynamics simulation output and the post-processing required to generate the input data for the chemistry code, using the UT-PROJ projection code for the aggregation service. UT-PROJ also provides the mapping function for the attribute space service, for mapping between the hydrodynamics grid and the chemistry grid. The role of T2 in coupling the hydrodynamics and chemical transport simulators is shown in Figure 4.

The hydrodynamics simulator simulates circulation patterns in coastal shelves and estuaries with an unstructured grid. The elements of the grid are two-dimensional triangles, or three-dimensional tetrahedra. The elevations and the velocities at all vertices of the grid are computed every time step, and must be written into T2. The attribute space for this data to be written into T2 is therefore a three-dimensional spatial grid, with an additional dimension for the simulation time. This attribute space must be registered with T2 as a base attribute space.

Data blocks are the basic unit for data storage and retrieval in T2. Thus, the data generated by the hydrodynamics module must be partitioned into blocks using an unstructured grid partitioning algorithm (e.g., recursive coordinate bisection [4]). To allow data blocks to be accessed by their spatio-temporal attributes, an index (e.g., an R-tree) is created, with a bounding box for each data block used as the access key. The bounding boxes are also used by the default T2 declustering and clustering algorithms in the data loading service, which determines the location on the disk farm to which each data block is assigned.

The hydrodynamics input required by the chemistry code, CE-QUAL-ICM, are the fluid depths, flow rates and turbulent eddy viscosity coefficients. CE-QUAL-ICM also uses a three-dimensional unstructured grid, but the grid consists of 3D hexahedral elements, as compared to the triangular elements used by the hydrodynamics code. Since, CE-QUAL-ICM is a finite-volume algorithm, the data is required at the midpoints of cell edges. Due to differences in the time scales of interest, the simulation time steps used by CE-QUAL-ICM are usually much larger than those used by hydrodynamics simulators such as ADCIRC and UTBEST. This implies that it is not sufficient to access one time step of the hydrodynamics data from T2, so the aggregation function in T2 must perform time averaging. In addition, because the unstructured grid used by CE-QUAL-ICM is not the same as the one used in the hydrodynamics code, the hydrodynamics data has to be projected onto the grid for CE-QUAL-ICM. Therefore the unstructured grid for CE-QUAL-ICM must also be registered as a base attribute space with the T2 attribute space service. The functions for performing the projection, both the T2 mapping function in the attribute space service and the aggregation function for the data aggregation service, are provided by the UT-PROJ code mentioned in Section 2.

A typical query from CE-QUAL-ICM specifies 1) the hydrodynamics grid of interest, 2) the chemistry grid that the output will be computed into, 3) the mapping function and aggregation function to use (e.g., from UT-PROJ), and 4) the time period of interest in the hydrodynamics simulation. No transformation function is currently required for this application. The query also specifies where to send the output data (wherever CE-QUAL-ICM is running). For this application, it is desirable to allow one query to specify multiple sets of time steps to be aggregated, because CE-QUAL-ICM will otherwise send multiple queries to T2 that are identical except for the time period specified. Again, as for the AVHRR database example, the internal T2 services (query interface, query planning and query execution) are then performed to generate the output grid data that is sent to the desired destination.

Because T2 and both the hydrodynamics and chemistry simulations are parallel programs, an efficient mechanism for moving large amounts of data between parallel programs is required. The data transport mechanism could be implemented directly by completely specifying the communication between all source and destination processes individually, or can be more conveniently performed using the Maryland Meta-Chaos library [10]. Meta-Chaos is a runtime library that has been developed to facilitate exchange of distributed data structures between two parallel programs, and generates the required communication between pairs of processes from a high-level specification.

6 Preliminary Experimental Results

Most of the T2 services have been implemented and a prototype is currently running on an IBM SP-2 at the University of Maryland. The Maryland SP-2 consists of 16 RS6000/390 processing nodes (so-called **thin** nodes), running AIX 4.2, with six fast disks (IBM Starfire 7200) attached to each processing node. The T2 prototype is implemented in C++ and uses inheritance to implement much of the customization interface. In the current implementation, the planning service is able to generate query plans using both the input partitioning and the output partitioning strategies discussed in Section 4.6, and we are in the process of developing cost models that would allow the planning service to choose the better strategy for a given query, based on both the resources available

global	North America	South America	Africa	Australia
1736	402	167	203	70

Table 1: Amount of data retrieved from disk for various sample queries, in MB.

	global	North America	South America	Africa	Australia
T2	134.3	26.3	9.8	12.8	4.3
Titan	128.5	25.3	9.1	12.2	4.3

Table 2: Query processing times (in seconds) for T2 and Titan.

in the parallel machine and the query processing information stored by the various services. For a query with an intermediate result (accumulator) that is too large to fit entirely in the memory of the available processors, the planning service currently partitions the intermediate result into smaller chunks and has all the processors cooperate to compute each chunk in turn. For portability, the T2 query execution service uses the POSIX lio_listio interface for its non-blocking I/O operations, and MPI [35] as its underlying interprocessor communication layer.

To show the effectiveness of T2, we have used the various T2 services to implement a remote-sensing image database that emulates the functionality of Titan [6] on the Maryland SP-2. Titan is a custom-built image database for storing remotely sensed data, and is currently operational on the Maryland SP-2, containing about 24 GB of data from the AVHRR sensor on the National Oceanic and Atmospheric Administration NOAA-7 satellite. Titan dedicates one of the sixteen SP-2 processing nodes as a front-end node, which interacts with Java GUI client programs, and uses the other fifteen nodes as the back-end processing and data retrieval nodes. Four disks on each of the back-end nodes are used to store the AVHRR data chunks, each consisting of 204×204 IFOVs, and the same declustering and clustering algorithms that are provided by T2 are used for placing the IFOV chunks onto the sixty disks. In prior work [6] we have shown that Titan delivers good performance for both small and large queries.

A set of sample queries were used to evaluate the performance of the T2 implementation against that of Titan. These are 10-day NDVI composite queries, as described in Section 5.1, and Table 1 shows the total amount of AVHRR data that needs to be retrieved from disk to satisfy the queries. As in T2, Titan is able to support both input partitioning and output partitioning strategies. Since interprocessor communication does not overlap well with the computation on the Maryland SP-2 [34], and the AVHRR data chunks are not uniformly distributed across the entire spatial extent of the database, we have chosen to show results for the input partitioning strategy, which performs better than the output partitioning strategy due to both lower communication overhead and better load balance. Figure 2 shows the query processing times for both Titan and the T2 implementation. The results show that the T2 performance is very close to the Titan performance, indicating that performance does not suffer much from the more general design of T2.

The final set of results shows the benefits of integrating data retrieval and processing in T2. Table 3 shows the disk read times for the sample queries, per processor. The I/O times are approximately the same on all processors, because the declustering algorithm does a good job of distributing queries across all disks.

The I/O times show part of the additional cost that will be incurred in a traditional database

global	North America	South America	Africa	Australia
11.4	2.6	1.2	1.2	0.6

Table 3: Disk read times (in seconds) for T2

architecture that does not allow complex application processing to be performed within the system running the multi-dimensional database system. In such an architecture, it could be much more difficult to overlap the I/O with computation. The I/O costs for the remote sensing database implemented in T2 are only about 10% of the total execution time for the sample queries, because this application performs a large amount of computation for every data chunk accessed by a query. However, other applications, such as the Virtual Microscope, perform much less computation per data chunk.

The advantages of integrated data retrieval and processing can also be appreciated by considering the potential cost of moving data from a database to a separate program that would carry out the necessary computations. For instance, in the global query depicted in Table 1, we see that 1.736 Gbytes would have to be moved from the database system to the application. In general, this data movement would involve either disk I/O or interprocessor communication. For instance, at a rate of 10Mbytes per second, a single 1.736 Gbyte data transfer would require 173.6 seconds, compared to the total T2 retrieval and processing cost of 134.3 seconds.

7 Current Status and Future Work

We have presented T2, a customizable parallel database that integrates storage, retrieval and processing of multi-dimensional datasets. We have described the various services provided by T2, and further shown how several of those services can be customized for a particular application. We have also provided preliminary performance results showing that an AVHRR database application implemented using the T2 services performs almost as well as a custom implementation of the same application.

We are currently in the process of optimizing the various T2 services, and are continuing to experiment with the planning algorithms and cost models for the query planning service. We are also working on generalizing the design of the various services to handle multiple simultaneous queries. In addition, we are beginning to implement the Virtual Microscope and the system for coupling multiple simulations for water contamination studies using T2. We are also starting to investigate techniques for extending T2 to tertiary storage, to efficiently store and process datasets that are too large to fit into secondary storage.

Finally, we are beginning to investigate how the T2 services for multi-dimensional data sets can be integrated with services provided by more general purpose databases (e.g., relational or object-relational commercial databases). Our goal in that research is to define standard ways of integrating T2 services into existing database systems. That would allow, for instance, the ability to do a join between data stored in a relational database and multi-dimensional data stored in T2, and perform the join efficiently.

Acknowledgements

We would like to thank Tahsin Kurc for his help in understanding the University of Texas bay and estuary simulation codes.

References

[1] A. Acharya, M. Uysal, R. Bennett, A. Mendelson, M. Beynon, J. Hollingsworth, J. Saltz, and A. Sussman. Tuning the performance of I/O-intensive parallel applications. In *Proceedings of the Fourth ACM Workshop on I/O in Parallel and Distributed Systems*, May 1996.

[2] P. Baumann, P. Furtado, R. Ritsch, and N. Widmann. Geo/environmental and medical data management in the RasDaMan system. In *Proceedings of the 23th VLDB Conference*, pages 548–552, Aug. 1997.

[3] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger. The R^*-tree: An efficient and robust access method for points and rectangles. In *Proceedings of the 1990 ACM-SIGMOD Conference*, pages 322–331, May 1990.

[4] M. Berger and S. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Transactions on Computers*, C-36(5):570–580, May 1987.

[5] C. F. Cerco and T. Cole. User's guide to the CE-QUAL-ICM three-dimensional eutrophication model, release version 1.0. Technical Report EL-95-15, US Army Corps of Engineers Water Experiment Station, Vicksburg, MS, 1995.

[6] C. Chang, B. Moon, A. Acharya, C. Shock, A. Sussman, and J. Saltz. Titan: A high performance remote-sensing database. In *Proceedings of the 1997 International Conference on Data Engineering*, pages 375–384. IEEE Computer Society Press, Apr. 1997.

[7] S. Chippada, C. N. Dawson, M. L. Martínez, and M. F. Wheeler. A Godunov-type finite volume method for the system of shallow water equations. *Computer Methods in Applied Mechanics and Engineering (to appear)*, 1997. Also a TICAM Report 96-57, University of Texas, Austin, TX 78712.

[8] D. J. DeWitt, N. Kabra, J. Luo, J. M. Patel, and J.-B. Yu. Client–server Paradise. In *Proceedings of the 20th VLDB Conference*, pages 558–569. Morgan Kaufmann Publishers, Inc., 1994.

[9] J. Dongarra, J. D. Croz, S. Hammarling, and I. S. Duff. A set of level 3 basic linear algebra subprograms. *ACM Transactions on Mathematical Software*, 16(1):1–17, Mar. 1990.

[10] G. Edjlali, A. Sussman, and J. Saltz. Interoperability of data parallel runtime libraries. In *Proceedings of the Eleventh International Parallel Processing Symposium*. IEEE Computer Society Press, Apr. 1997.

[11] M. T. Fang, R. C. T. Lee, and C. C. Chang. The idea of de-clustering and its applications. In *Proceedings of the 12th VLDB Conference*, pages 181–188, 1986.

[12] R. Ferreira, B. Moon, J. Humphries, A. Sussman, J. Saltz, R. Miller, and A. Demarzo. The Virtual Microscope. In *Proceedings of the 1997 AMIA Annual Fall Symposium*, pages 449–453. American Medical Informatics Association, Hanley and Belfus, Inc., Oct. 1997. Also available as University of Maryland Technical Report CS-TR-3777 and UMIACS-TR-97-35.

[13] R. A. Finkel and J. L. Bentley. Quad-Trees - a data structure for retrieval on composite keys. *Acta Informatica*, 4:1–9, 1974.

[14] NSF/ARPA Grand Challenge Project at the University of Maryland for Land Cover Dynamics, 1995. *http://www.umiacs.umd.edu:80/research/GC/*.

[15] J. Gray, A. Bosworth, A. Layman, and H. Pirahesh. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proceedings of the 1996 International Conference on Data Engineering*, pages 152–159, Feb. 1996.

[16] A. Guttman. R-Trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM-SIGMOD Conference*, pages 47–57, June 1984.

[17] Y. Ioannidis, M. Livny, S. Gupta, and N. Ponnekanti. ZOO: A desktop experiment management environment. In *Proc. 22nd International VLDB Conference*, pages 274–85, 1996.

[18] C. O. Justice, J. R. G. Townshend, B. N. Holben, and C. J. Tucker. Analysis of the phenology of global vegetation using meteorological satellite data. *International Journal of Remote Sensing*, pages 1271–1318, 1985.

[19] I. Kodukula, N. Ahmed, and K. Pingali. Data-centric multi-level blocking. In *Proceedings of the SIGPLAN '97 Conference on Programming Language Design and Implementation*, pages 346–357. ACM Press, June 1997. ACM SIGPLAN Notices, Vol. 32, No. 5.

[20] Land Satellite Thematic Mapper (TM). *http://edcwww.cr.usgs.gov/nsdi/html/landsat_tm/ landsat_tm.*

[21] S. Liang, L. Davis, J. Townshend, R. Chellappa, R. Dubayah, S. Goward, J. JaJa, S. Krishnamachari, N. Roussopoulos, J. Saltz, H. Samet, T. Shock, and M. Srinivasan. Land cover dynamics investigation using parallel computers. In *Proceedings of the 1995 International Geoscience and Remote Sensing Symposium, Quantitative Remote Sensing for Science and Applications.*, pages 332–4, July 1995.

[22] M. Livny, R. Ramakrishnan, K. Beyer, G. Chen, D. Donjerkovic, S. Lawande, J. Myllymaki, and K. Wenger. DEVise: integrated querying and visual exploration of large datasets. In *Proceedings of ACM SIGMOD*, pages 301–12, 1997.

[23] R. A. Luettich, J. J. Westerink, and N. W. Scheffner. *ADCIRC*: An advanced three-dimensional circulation model for shelves, coasts, and estuaries. Technical Report 1, Department of the Army, U.S. Army Corps of Engineers, Washington, D.C. 20314-1000, December 1991.

[24] K.-L. Ma and Z. Zheng. 3D visualization of unsteady 2D airplane wake vortices. In *Proceedings of Visualization'94*, pages 124–31, Oct 1994.

[25] The Moderate Resolution Imaging Spectrometer. *http://ltpwww.gsfc.nasa.gov/MODIS/MODIS.html.*

[26] B. Moon, A. Acharya, and J. Saltz. Study of scalable declustering algorithms for parallel grid files. In *Proceedings of the Tenth International Parallel Processing Symposium*, pages 434–440. IEEE Computer Society Press, Apr. 1996.

[27] B. Moon and J. H. Saltz. Scalability analysis of declustering methods for multidimensional range queries. *IEEE Transactions on Knowledge and Data Engineering*, 1997. To appear.

[28] T. C. Mowry, M. S. Lam, and A. Gupta. Design and evaluation of a compiler algorithm for prefetching. In *Proceedings of the Fifth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS V)*, pages 62–73. ACM Press, Oct. 1992.

[29] NASA Goddard Distributed Active Archive Center (DAAC). Advanced Very High Resolution Radiometer Global Area Coverage (AVHRR GAC) data. *http://daac.gsfc.nasa.gov/CAMPAIGN_DOCS/ LAND_BIO/origins.html.*

[30] S. Olsson and C. Busch. A national telepathology trial in Sweden: Feasibility and assessment. *Arch. Anat. Cytol. Pathol.*, 43:234–241, 1995.

[31] The Oracle 8 spatial data cartridge, 1997. *http://www.oracle.com/st/cartridges/spatial/.*

[32] J. Patel et al. Building a scaleable geo-spatial DBMS: technology, implementation, and evaluation. In *Proceedings of ACM SIGMOD*, pages 336–47, 1997.

[33] G. Patnaik, K. Kailasnath, and E. Oran. Effect of gravity on flame instabilities in premixed gases. *AIAA Journal*, 29(12):2141–8, Dec 1991.

[34] M. Snir, P. Hochschild, D. Frye, and K. Gildea. The communication software and parallel environment of the IBM SP2. *IBM Systems Journal*, 34(2):205–221, 1995.

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

[36] SpatialWare DataBlade Module (from MapInfo) Corp, 1997. *http://www.informix.com/informix/ bussol/iusdb/databld/dbtech/sheets/spware.htm.*

[37] D. R. Steinwand. Mapping raster imagery to the Interrupted Goodes Homolosine projection. *http://edcwww.cr.usgs.gov/landdaac/1KM/goodesarticle.html.*

[38] M. Stonebraker. An overview of the Sequoia 2000 project. In *Proceedings of the 1992 COMPCON Conference*, pages 383–388, 1992.

[39] T. Tanaka. Configurations of the solar wind flow and magnetic field around the planets with no magnetic field: calculation by a new MHD. *Jounal of Geophysical Research*, 98(A10):17251–62, Oct 1993.

[40] The USGS General Cartographic Transformation Package, version 2.0.2. *ftp://mapping.usgs.gov/ pub/software/current_software/gctp/*, 1997.

[41] M. Weinstein and J. I. Epstein. Telepathology diagnosis of prostate needle biopsies. *Human Pathology*, 28(1):22–29, Jan. 1997.

[42] R. S. Weinstein, A. Bhattacharyya, A. R. Graham, and J. R. Davis. Telepathology: A ten-year progress report. *Human Pathology*, 28(1):1–7, Jan. 1997.