FINAL REPORT

For Grant# DE-FG05-92ER25143

entitled

"Partnership in Computational Science"

Submitted to

Department of Energy
Oak Ridge Operations Office
P.O. Box 2001
Oak Ridge, TN 37831

By

Richard E. Ewing, Ph.D.
Institute for Scientific Computation
Texas A&M University
612 Blocker
College Station, TX 77843-3404

DISCLAIMER

DISCLAIMER

Portions of this document may be illegible electronic image products. Images are produced from the best available original document.

Partnership in Computational Science

The funding in this project was used to purchase a Sigma Model 2 INTEL distributed memory computer and 3 years of maintenance support from INTEL. This computer system was used in fulfilling the milestones of the Partnership in Computational Science (PICS) consortium that was also funded by the Department of Energy through Oak Ridge National Laboratory (ORNL). The computer delivered had the minimum configuration of nodes, memory, and disk necessary to provide the scalable development capabilities to reach the teraflop goals of the PICS consortium for the High Performance Computing Capabilities Initiative.

As part of the PICS grant through ORNL, the research group at Texas A&M University was charged with the development of the flow code to form the basis for a new state-of-the-art multiphase and multicomponent contaminant transport model. The development of this model required extensive computational capabilities and the final code was to run effectively on the INTEL Paragon computer. Earlier versions of this code were tried on the INTEL computer at ORNL via the internet, however the extensive testing necessary to debug a complex code on a massively distributed memory computer could not be performed adequately over the network. By having a comparable computer system on-site, the testing necessary to debug the codes was performed more efficiently and effectively.

In order to complete the milestones and deliverables of the PICS consortium, Texas A&M University researchers needed to combine codes they designed with codes designed at other universities. The Sigma Model 2 computer purchased by Texas A&M University was the same configuration as the machines purchased by those universities. This insured hardware compatibility during the development phase and eliminated congruity problems when the codes were combined. This computer was also the same basic architecture as the Paragon located at ORNL and thus provided a scalable version of the computer intended to sustain teraflop capabilities.

The code development that was accomplished at Texas A&M University through the computational capabilities of the Sigma Model 2 computer system is explained in more detail in Chapter 3 of the report entitled "Grand Challenge Problems in Environmental Modeling and Remediation: Groundwater Contaminant Transport Final Project Report 1998" submitted to the U.S. Department of Energy under contract # DE-AC05-96-OR22464.

Introduction

This is a final report on the OPTIMASS project which is a collaborative research effort between researchers at the Lawrence Berkeley Laboratory, the Lawrence Livermore Laboratory, and the University of Maryland.

This report is organized as follows. Section 1 describes the background for the project. Section 2 discusses our approach which makes use of abstracts as a data reduction method. After an initial definition of the method, we outline the ATree data structure which is the representation that we have developed for handling the abstracts. Sections 3, 4, and 5 contains an outline of our plans and partial accomplishments for the current year and the next two fiscal years of the project.

1. Background

Large-scale scientific simulations, experiments, and observational projects, generate large multidimensional datasets and then store them temporarily or permanently in an archival mass storage system (MSS) until it is required to retrieve them for analysis or visualization. For example, a single dataset (usually a collection of time-history output) from a climate model simulation may produce from one to twenty gigabytes of data. Typically, this dataset is stored on up to one hundred magnetic tapes, cartridges, or optical disks. These kinds of tertiary devices (i.e., one level below magnetic disk), even if robotically controlled, are relatively slow. Taking into account the time it takes to load, search, read, rewind, and unload a large number of cartridges, it can take many hours to retrieve a subset of interest from a large dataset.

An important aspect of a scientific investigation is to efficiently access the relevant subsets of information contained within much larger datasets for analysis and interactive visualization. Naturally, the data access depends on the method used for the initial storage of this dataset. Because a dataset is typically stored on tertiary storage systems in the order it is produced and not by the order in which it will be retrieved, a large portion of the dataset needs to be read in order to extract the desired subset. This leads to long delays (30 minutes to several hours is common) depending on the size of the dataset, the speed of the device used, and the usage load on the mass storage system.

The main concept we pursue here is that datasets should be organized on tertiary storage reflecting the way they are going to be accessed (i.e. anticipated queries) rather than the way they were generated or collected. In order to have an effective use of the tertiary storage we need to enhance current storage server protocols to permit control over physical placement of data on the storage devices. In addition, these protocols need to be enhanced to support multiple file reads in a single request.

In order to have a practical and realistic environment, we choose to focus on developing efficient storage and retrieval of climate modeling data generated by the Program for Climate Model Diagnosis and Intercomparison (PCMDI). PCMDI was established at Lawrence Livermore National Laboratory (LLNL) to mount a sustained program of analysis and experimentation with climate models, in cooperation with the international climate modeling community [1]. To date, PCMDI has generated over one terabyte of data, mainly consisting of very large, spatio-temporal, multidimensional data arrays.

A similar situation exists with many scientific application areas. For example, the Earth Observing System (EOS) currently being developed by NASA [2], is expected to produce very large datasets (100s of gigabytes each). The total amount of data that will be generated

is expected to reach several petabytes, and thus will reside mostly on tertiary storage devices. Such datasets are usually abstracted into so called "browse sets" that are small enough to be stored on disk (using coarser granularity and/or summarization, such as monthly averages). Users typically explore the browse sets at first, and eventually focus on a subset of the dataset they are interested in. We address in this proposal this last step of extracting the desired subsets from datasets that are large enough to be typically stored on tape.

Future hardware technology developments will certainly help the situation. Data transfer rates are likely to increase by as much as an order of magnitude as will tape and cartridge capacities. However, new supercomputers and massively parallel processor technologies will outstrip this capacity by allowing scientists to calculate ever finer resolutions and more time steps, and thus generating much more data. Because most of the data generated by models and experiments will still be required to reside on tertiary devices, and because it will usually be the case that only a subset of that data is of immediate interest, effective management of very large scientific datasets will be an ongoing concern. However, there is an additional benefit to our approach. Even if we accept the premise that users will be tolerant of long delays (i.e. placing orders that take several hours or overnight to fill), it is still in the best interest of mass storage facilities to be able to process requests more efficiently, by avoiding the reading of data that are not needed. This translates into savings on the hardware needed to support an average access load.

It is not realistic to expect commercial database systems to add efficient support for various types of tertiary storage soon. But even if such capabilities existed, we advocate an approach that the mass storage service should be outside the data management system, and that various software systems (including future data management systems) will interface to this service through a standardized protocol. The IEEE is actively pursuing such standard protocols [3] and many commercially available storage system vendors have stated that they will help develop and support this standards effort for a variety of tertiary devices. Another advantage to our approach is that existing software applications, such as analysis and visualization software, can interface directly to the mass storage service. For efficiency reasons, many applications use specialized internal data formats and often prefer to interface to files directly rather than use a data management system.

2. Approach

Our goal is to read as little data as possible from the MSS in order to satisfy the subset request. For example, for visualization studies the most likely access pattern is regional (in terms of spatial coordinates) over extended time periods. For such applications, the dataset should be partitioned and stored as regional "bins" or "clusters" over time, as opposed to the traditional way of storing data globally for one time slice. In general, the portions of a dataset that satisfy a query may be scattered over different parts of the dataset, or even on multiple volumes. For example, typical climate simulation programs generate multiple files, each for a period of 5 days for all variables of the dataset. Thus, for a query that requests a single variable (say "precipitation") for a specific month at ground level, the relevant parts of the dataset reside on 6 files (each for a 5 day period). These files may be stored on multiple volumes. Further, only a subset of each file is needed since we are only interested in a single variable and only at ground level. If we collected all the parts relevant to a query and put them into a single file, then we would have the ideal cluster for that query. Of course, the problem is one of striking a balance between the requirements of all queries, and designing clusters that will be as close as possible to the ideal cluster of each query. This idea is a common methodology used in data management systems (called "physical database design"). However, such methods have not been applied or investigated much in the context of mass storage systems.

Using abstracts as a data reduction method.

In addition to partitioning methods for reducing the amount of data that need to be extracted for analysis, we are investigating a storage management technique called abstracts. Abstracts are extraction of a certain ``essential" part of the original data. Abstracts can be seen as precomputed answers to an anticipated query, or precomputed data that helps answer a class of queries. Abstracts are several orders of magnitude smaller than the original data sets.

A data set can have multiple abstracts associated with it, each corresponding to different ``essential" part of data set. Thus, each abstract is a different view of data. What the abstracts really are depends on the data and on the *context* in which the data is used. Note that an abstract need not produce the same kind of information as does the original data.

Here we distinguish two kinds of abstracts, based on the type of data that an abstract represents with respect to the original data:

Abstracts which summarize the data. This kind of abstract takes advantage of the fact that a scientist may want to sacrifice some accuracy for having an interactive response of the system. Here, the abstract may simply be a low-resolution version of the original data (e.g., monthly average: only time coordinate is at low-resolution; or using T21 data instead of T106, etc.). Also, an abstract may be a compressed version of the original data using either lossless or lossy compression. (Usually, lossy compression is employed since the reduction ratios achieved by non-lossy compression are not enough. However, non-lossy compression may be used to store the original data set.)

For example, it is very common to visualize (animate) surface temperatures over a time period. For this, a discrete cosine transform, followed by quantization, and entropy encoding at display resolution may be appropriate abstraction method. In general, the type of compression technique applied is specific to the type of data involved.

To sum up, these type of abstracts sacrifice accuracy for performance since an inexact approximation is used. Some of the methods that might be employed are:
Resolution reduction.
Quantization.
Transformation into another domain (e.g., DCT, CEOF, or wavelet transformation of data).
Lossy compression (employing any of the methods above).

Abstracts which index the contents of the data. These type of abstracts facilitate quick answers to queries such as ``find all images which contain a hurricane", or ``find all anomalies in the data". The abstract essentially is an index into the original data or into another summary abstract on the result of a *classification function* (or *abstracting function*).

This is *instance (content) indexing* as opposed to traditional value indexing. A classification function can have an imprecise (fuzzy) classification. In the example above, the abstraction function is a function which returns spatio-temporal regions that contain a hurricane (or anomalies).

In order to facilitate instance indexing, a classification function is run on one or more variables in the dataset. The function returns a list of regions which contain the feature in

question. Next, the description of those regions, along with other attributes that characterize the feature are stored in an appropriate data structure. Data structures that might be effectively used are PMR-trees and R-trees, and their variants [Samet90].

Benefits of abstracts are many. If a query can be answered from the abstract, then the I/O bandwidth increases, and the response time decreases. Since abstracts are several orders of magnitude smaller than the original data set, abstracts can be kept on secondary storage, instead on a tertiary storage. Hence, the speedup is cumulative (smaller size + faster storage device). This speedup might be even greater since now caching can be effectively employed.

Abstracts should be generated at the time when data is brought into the system, since intended uses and anticipated access patterns of data are more or less known. If the query cannot be answered from the abstract, the we have to access the original data and run a query over the full-size data set. In this case, it is desirable to be able to generate abstracts in an ad-hoc manner. Generating an abstract is equivalent to going through all of the data (which we have to do anyway in the case of an un-anticipated query), and thus we might as well generate an abstract along the way.

The use of a tree structure (A-Tree) to represent abstracts

ATree: The Engine

We have developed and implemented a storage access method called an *ATree* suitable for storage of abstracts of very large multidimensional vector field data. The ATree data structure is complementary to the "dimension-based" partitioning described above---that is, theATree provides "feature based" partitioning of space. Spatial subset queries can efficiently be answered using the dimension-based methods. However, to answer a feature query (e.g., "find all regions which contain a hurricane"), it may be necessary to search the entire dataset. The ATree data structure allows this type of queries to be efficiently answered.

The ATree data structure is an extensible data structure with the following main features:

- The ATree is a multidimensional indexing data structure. A spatial index allows quick access to data based on its spatial location.
- Data is partitioned into chunks according to the principle of spatial locality. The size and shape of the chunks is chosen according to the expected usage of the data.
- Data chunks are organized (laid out) on archival storage according to the expected usage of the data. Currently, Morton codes are used to specify the layout, so that data chunks are stored on secondary/tertiary storage devices according to the principle of spatial locality.
- The ATree permits for hiearachial compression techniques to be applied (both lossy and non-lossy). Note that unlike traditional compression techniques, the

ATree permits for direct indexing into compressed data. This means that only the chunks of data that are accessed by user are decompressed.

- The ATree can store multi-resolution data.

- The ATree permits for non-spatial information to be included which facilitates quick retrieval of data based on non-spatial attributes.

The ATree data structure is an extension of the region quadtree data structure. We describe the ATree data structure below:

Subdivision Scheme

Unlike the region quadtree which always subdivides quadrants into exactly 4 disjoint son quadrants, the region nodes of the ATree, called orthants, are subdivided along pre-specified coordinates at each level of the tree. The same coordinates are subdivided for all the orthants at the same level. At least one coordinate has to be subdivided at each level of decomposition. When an orthant is subdivided along a coordinate, it is subdivided into two halves.

For example, we can implement KD-trees by defining subdivision of orthants to be performed on different coordinate (cyclically) at each level. Traditional quadtree decomposition can be achieved by ATree when orthants are subdivided along all coordinates at each level.

The benefits of the ATree in comparision to the region quadtree approach are as follows:

- Region quadtrees are not suitable for storing data with non-square extents (blocks) since the leaf nodes also have non-square extents, and in the case when one coordinate has many more samples than the others, this is not acceptable. The subdivision scheme allows data with non-square extents to be stored in ATree.

- Multidimensional generalizations of quadtrees are impractical even for moderately high dimensions due to the high fan-out factor (2^d). The subdivision criterion provides fine control over the fan-out factor at each level of the ATree.

- Orthants specify how data is chunked. By controlling the subdivision scheme, we can control the shape of data chunks.

Location Codes

In order to linearize multidimensional data, a location code scheme is used. The linearization process is a function lc that maps hyperrectangles to integers such that k=lc(r) where r is a hyperrectangle corresponding to some orthant in the

subdivision, and k is the orthant's number. The function lc has to be 1-1 and onto since lc^{-1} has to be defined.

Currently, we are using location codes based on the Morton order. The Morton order (also known as Z-order) linearizes multidimensional space so that the orthants which are spatially close tend to have their location code values close as well.

The ATree data structure allows for any consistent location code scheme to be used. By using different location code schemes we can specify how orthants (data chunks) are laid out on archival storage.

Transformation Function

A transformation functions specifies what is actually stored with each ATree orthant node. It is applied to both internal and leaf nodes of the tree. By storing data in non-leaf orthant nodes we get the pyramid variant of the ATree data structure. This allows data to be stored at different resolutions so that orthants near the root of the tree contain low resolution data, and as we descend the tree, data is refined further.

Note that in order to facilitate data retrieval , the inverse of the transformation function must exist. Let $d=t(o)$, where o is the subset of original data corresponding to some orthant, t is a transformation function, and d is the data returned by the transformation function, i.e., data that is actually stored on disk. Now, let t^{-1} be the inverse of the transformation function, that is, the function that takes raw data retrieved from the disk and returns data *corresponding* to the original untransformed data. Ideally, $o=t^{-1}(t(o))$ but this needs not to be the case, such as, for example, in the case of lossy compression. The transformation function can also specify non-spatial data that is to be stored with each orthant. This information can be used to aid non-spatial queries.

Subdivision Criterion

The subdivision criterion specifies how deeply we go in subdividing orthants. If the ATree does not store internal nodes and if the transformation function is the identity function, the then subdivision criterion effectively specifies the size of data chunks stored on disk. In this case, one of the approaches that we employ is to stop subdividing space as soon as data associated with the orthant is of certain size (leaf-orthant-data-size). In order to maximize performance when data is stored on secondary storage devices, we are using virtual memory page size as leaf-orthant-data-size. For tertiary devices, bigger chunks are appropriate, e.g., we may use platter size.

User Interface

The ATree data structure is implemented as an object library. Currently, the ATree uses the B-Tree as underlying storage mechanism.

We have implemented a suite of AVS modules to operate on ATrees. AVS employs boxes-and-arrows visual programing paradigm. Each AVS module represents some high level operation and is graphically represented by a box. Functionally, a module is a Unix process with a number of input and output arguments, called input and output ports. An output port of one module can be connected to an input port of another module, provided that the port types match. This connection is graphically represented by a colored line (arrow) between modules. The line represents data flowing from the first module into the second, and the color of the line represents its data type.

Given the size of the ATree data sets, it is not possible to transmit a whole data set from one module to another. Rather, the following approach is taken. The ATree object library implements ATree data structure as permanent objects (in OO sense). Only a reference to a permanent ATree object is transmitted between two modules, so there is very little overhead in communication between modules.

We have implemented the following modules:

Build. Input to this module is a raw data set. The output from this module is the reference to the new ATree object. In addition, this module takes a number of parameters which specify how the resulting ATree is built. These parameters include Subdivision Scheme, Split Criterion, Transformation Function, and Location Code object specifications.

Subset. This module is a basic spatial query. The input to this module is a hyperrectangle, and the output is a data set corresponding to the specified spatial domain.

There are two versions of this module. The first version outputs another ATree as the result of this operation, and the second version outputs multidimensional field in standard AVS format. This field can be then piped into any other AVS module that takes field data as input for further analysis and visualization.

Composite. This module is a spatial join operation. The input is two ATrees which are combined into a single data sets: $d_3(x)=f(d_1(x),d_2(x))$ where d_1 and d_2 are input data sets, d_3 is an output data set, $x \in \text{domain}(d_1) \cup \text{domain}(d_2)$, where f is the composite function.

There are two versions of this module. The first version outputs another ATree. The second version outputs an AVS field.

3. Technical Progress FY95

The use of a tree structure (A-Tree) to represent abstracts

1 Bottom-up build algorithm.

The top-down build algorithm requires all of the data to be available at build time. However, this may be impossible for very large data sets. In order to deal with large data sets, a bottom-up algorithm is designed and implemented. A bottom-up algorithm grows the tree starting from the leaf nodes, and merges the orthant nodes as necessary.

The bottom-up algorithm works in two phases. In the first phase, data is read from the data source (e.g., secondary or tertiary storage) in one pass. The bottommost leaf nodes of the tree are generated. In the second phase, the leaf nodes are merged as necessary, possibly all the way up to the root node. The merge process is guided by the subdivision criterion function.

2 Hybrid build algorithm.

Note, that not every transform function is appropriate for the bottom-up build algorithm. For example, a transform function that needs to know all the original data associated with an orthant node will require that we have all the data available for the root node as well. In this case, bottom-up algorithm gives us no advantage over a top-down algorithm.

In general, the bottom-up approach will be superior for trees where no data is associated with nodes close to the root, or trees whose orthant node data close to root can be derived from immediate children data.

One possible approach is to use a hybrid build algorithm. This algorithm first builds top-down smaller ATrees either in core or in secondary memory. Next, the algorithm proceeds by merging these smaller ATrees into an ATree representing the entire data set. This merging process is performed bottom-up.

3 Object-oriented implementation and performance improvements.

The ATree implementation has greatly been cleaned up. Most of the classes have been reimplemented to allow more dynamic behavior and greater ease of use. Performance has been increased by using a faster and more robust B-tree code as the underlying storage mechanism.

An interactive interpreter is included as a part of the system. The interpreter allows automating interaction with the ATree system by the means of scripts. Objects (such as ATree, transformation, or subdivision objects) can be created and accessed at runtime. In addition, since the interpreter is fully object-oriented, it allows classes to be defined at runtime.

4. Plans for FY 1996

The use of a tree structure (A-Tree) to represent abstracts

1 Value search (feature queries).

Value (content) search algorithm represents a non-spatial query on the ATree data structure. It can be as simple as "find all regions where temperature is below 10°C" or as complex as "find all regions which contain a hurricane" (which is in fact a pattern recognition problem).

The assumption here is that there exists a function which we can apply to data in a spatial region which can give us an answer if the query is satisfied in that region or not. This answer need not be Boolean; it can be a number between 0 and 1 representing how certain we are that the region satisfies the query.

The ATree data structure can aid this kind of queries with the non-spatial information stored with orthant nodes. Further, the fact that this information can be stored with internal nodes as well can facilitate pruning of the search space.

2 Integration with the "dimension-based" partitioning module

The ATree data structure is complementary to the "dimension-based" partitioning. The ATree provides "feature based" partitioning of space. Spatial subset queries can efficiently be answered using the dimension-based methods. On the other hand, the ATree can be used to efficiently answer feature queries. These two systems need to be integrated together. When a query is posed to the system, the query is decomposed into a spatial and a non-spatial component. The spatial component is processed by the Storage Manager (Figure 2). This results in a set of requirements from the assembly module. On the other hand, the non-spatial (feature) query is processed by ATree. As a result another set of requirements from the assembly module is generated. These two sets of requirements then need to be intersected and the resulting set is used to retrieve the actual data.

3 Compression.

We intend to implement and experiment with various compression techniques. Currently only simple compression techniques such as quantization and resolution reduction are implemented. Some of the compression techniques of interest include DCT, CEOF, and wavelet transforms. These are the techniques that are often used by scientists to analyze the data. By using these techniques we not only reduce the size of the data set, but also effectively store pre-processed data.

5. Plans for FY 1997

The use of a tree structure (A-Tree) to represent abstracts

1 Parallel algorithms.

All three build algorithms (top-down, bottom-up, and hybrid) are suitable for parallelization. Parallel versions of these algoritm will allow us to deal with even larger data sets more efficiently. We intend to design and implement parallel versions of these three build algorithms. In addition, all the queries on the ATree data structure (both region-based and value-based queries), are also suitable for parallelization. Each orthant node can be assigned

to a different processor. This approach would significantly improve CPU intensive queries, such as queries on an ATree containing compressed data, and all value-based queries.

2 Accuracy for value-based queries.

Data stored in the ATree data structure is not exact (we are storing abstracted data sets). We intend to store accuracy information with each orthant node (both internal and leaf). This information specifies how accurate the abstracted data stored with the orthant is with respect to the original data. For value-based queries, the feature recognition function returns a number between 0 and 1 which represents certainity that the feature exists in the specified spatio-temporal region. We will use this value in conjunction with the accuracy value stored with each orthant node to prune low probability query results, as well as to indicate the accuracy of the final answer to the user.

References

[1] Gates, W., Potter, G., Phillips, T., Cess, R., An Overview of Ongoing Studies in Climate Model Diagnosis and Intercomparison, Energy Sciences Supercomputing 1990, UCRL-53916, pages 14-18.

[2] EOS Reference Handbook, NASA document NP-202, March 1993.

[3] Coleman, S., Miller, S., Eds., Mass Storage System Reference Model, Version 4, IEEE Technical Committee on Mass Storage Systems and Technology, May 1990.

[4] L.T. Chen, R. Drach, M. Keating, S. Louis, D. Rotem, A. Shoshani, "Efficient Organization and Access of Multi-Dimensional Datasets on Tertiary Storage Systems", To appear in a special issue on Scientific Databases, Information Systems Journal, Pergammon Press, Spring 1995.

[5] L.T. Chen, R. Drach, M. Keating, S. Louis, D. Rotem, A. Shoshani, Optimizing Tertiary Storage Organization and Access for Spatio-Temporal Datasets, Goddard Conference on Mass Storage Systems and Techno;ogies, Maryland, March, 1995.

[6] R. Coyne, and R. Watson, The National Storage Laboratory: Overview and Status, *Proc. Thirteenth IEEE Symposium on Mass Storage Systems*, Annecy, France, June 13-16, 1994.

[7] D. Teaff, R. Coyne, and R. Watson, The Architecture of the High Performance Storage System, *Fourth NASA GSFC Conference on Mass Storage Systems and Technologies*, College Park, MD, March 28-30, 1995.

[8] R. Drach, and R. Mobley, *DRS User's Guide,* UCRL-MA-110369, LLNL, March, 1994.

[9] S. Louis, and D. Teaff, Class of Service in the High Performance Storage System, *Second International Conference on Open Distributed Processing*, Brisbane, Australia, February 21-24, 1995.

[10] A. Segev, S. Seshadri, D. Rotem, Performance Evaluation of Mass Storage Systems for Scientific Databases, LBL report LBL-36092, 1994.