Algebraic Optimization of Computations over Scientific Databases

Richard Wolniewicz, University of Colorado at Boulder
Goetz Graefe, Portland State University

Abstract

Although scientific data analysis increasingly requires access and manipulation of large quantities of data, current database technology fails to meet the needs of scientific processing in a number of areas. To overcome acceptance problems among scientific users, database systems must provide performance and functionality comparable to current combinations of scientific programs and file systems. Therefore, we propose extending the concept of a database query to include numeric computation over scientific databases.

In this paper, we examine the specification of an integrated algebra that includes traditional database operators for pattern matching and search as well as numeric operators for scientific data sets. Through the use of a single integrated algebra, we can perform automatic optimization on scientific computations, realizing all of the traditional benefits of optimization.

We have experimented with a prototype optimizer which integrates sets, time series and spectra data types and operators on those types. Our results demonstrate that scientific database computations using numeric operators on multiple data types can be effectively optimized and permit performance gains that could not be realized without the integration.

This research has been performed in collaboration with the Space Grant College at the University of Colorado at Boulder, where the results are being applied to the analysis of experimental data from satellite observations.

1. Introduction

Modern scientific experiments in many fields, from biology to space science, are generating and analyzing large amounts of data. This necessitates greater consideration of data management facilities, in addition to the actual analysis of the experimental data. Therefore it is natural to look to existing database management technology to provide tools for managing scientific data.

Proceedings of the 19th VLDB Conference, Dublin, Ireland, 1993.

Many of the operations that must be performed on scientific data fall under the purview of traditional database systems. The growing volume of data available to scientists creates the need for facilities to correlate multiple data sets, including data from multiple experiments and historical data from past research [5, 28]. While database systems do provide facilities to manage large data volumes, current database systems do not provide support for the numeric computations required to perform correlations of scientific data. This shortcoming limits the use of database systems by scientific users.

The focus of this research is to extend the concept of a database query to encompass numerical computations over scientific databases, thereby creating an integrated algebra combining traditional database operators for pattern matching with scientific and numeric operators. Through this integration into a single, combined algebra, we gain the ability to perform automatic optimization of entire computations, with the resulting benefits of query optimization, algorithm selection and data independence becoming available to computations on scientific databases. This integration is illustrated in Figure 1, where the removal of the barrier between the database system and the computation allows the database optimizer to manipulate a larger portion of the application.

To begin our research into this integration, we have examined the addition of time series and spectra to the Volcano extensible query processing system [6-8]. The time series is a common data type for scientific computations, as it is often the basic format in which experimental data is retrieved. Similarly, the analysis of time series data is often performed in Fourier- (spectral) space, making time series and spectrum types central for the support of many scientific computations. These aspects make time series and spectra natural starting points for research into the optimization of scientific computations. Additionally, the variety of alternative computational techniques between spectral and time series methods offer an opportunity to examine complex

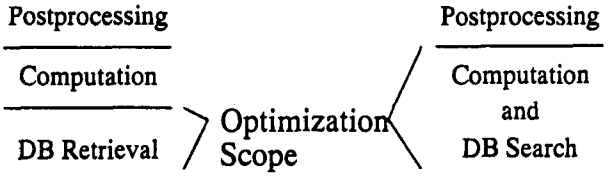


Figure 1: Removing the barrier between data management and numeric computation

transformations of numerical computations by an algebraic optimizer. The results learned from the examination of time series can then be used to extend this research to transformation of computations on other scientific data types, particularly multi-dimensional arrays.

Optimization has provided significant performance improvements in existing database systems. Through examination of operators on time series and spectra, we show that many of the algebraic optimization techniques used in current database optimizers can be extended to scientific computations, enabling the same performance improvements available through optimization in traditional database systems. Moreover, optimization frees the user from concerns about ordering of computation steps, algorithm selection, use of indices and other physical properties such as data distribution. Therefore optimization provides data independence to the user, a benefit which is as useful to the scientist as to any other database user. We believe optimization of integrated algebras including numeric operators is an important element for supporting scientific database users.

As part of this research, we have developed a prototype scientific optimizer using the Volcano optimizer generator [1, 7, 8]. This optimizer manipulates expressions over the integrated algebra, performing both logical transformations and physical algorithm selection. The optimizer takes advantage of physical properties of data objects, such as sort order and storage format, and performs resource planning. Since the greatest potential for optimization of computations arises during loop processing, i.e., processing of large bulk types in the database such as sets and arrays, these portions of the computation are the focus of the optimizer.

In Section 2, we discuss algebraic optimization in general. Section 3 discusses the data types supported by our prototype optimizer, and Section 4 presents the logical and physical operators manipulated by the prototype. Sections 5 and 6 address the logical transformations and implementation rules used by the optimizer. Section 7 presents two example computations that illustrate the scope and power of the prototype optimizer. Related work is discussed in Section 8, and our conclusions are presented in Section 9.

2. Algebraic Query Optimization

In this section we present an overview of transformation based algebraic query optimization, and show how the optimization of scientific computations fits into this framework.

To perform optimization of a computation over a scientific database system, the optimizer is given an expression consisting of logical operators on bulk data types. The bulk types we consider are sets (relations), time series, and spectra. Time series and spectra are used to illustrate the extension of the type system to scientific computations. Each operator in the computation takes one or more instances of a bulk type as its input, and produces a new instance of a bulk type as its output. The initial computation provided to the optimizer by the user does not include physical issues such as the availability of indices, the sort order of sets or the choice of specific algorithms for evaluation of operations. As these physical issues do not influence the result of the computation on the logical level, it is advantageous to hide these details from the user whenever possible. This provides data independence as in relational systems. Unlike relational optimizers, the model we are considering must address the need for multiple bulk types, thereby requiring that the initial computation be type-safe, i.e., that appropriate type coercion operators are included in the computation. In general, this can be verified as part of a preprocessing step. In many cases it is also possible to automatically add appropriate type coercion operators to a user computation that is not type-safe. The issues involved in such a preprocessor are not addressed in this paper. Rather, we assume that the initial computation presented to the optimizer is type-safe.

Starting with the initial computation, the optimizer repeatedly applies transformation rules to the logical expression to generate equivalent forms of that expression. These transformation rules translate a portion of an expression into a different form which will produce an equivalent result. Join associativity is an example of such a transformation in a relational query, as is performing filtering in either time- or Fourier-space in scientific computations. In the optimization of scientific computations, the identification of suitable transformation rules is of central importance. Unique problems must be considered, including, the equivalence of two numeric expressions given the issues of numerical accuracy and stability. These issues are addressed in more detail below.

Once applicable transformation rules have been used to generate equivalent logical expressions, the optimizer must find a set of physical algorithms that can implement or execute each expression. This process proceeds through the application of implementation rules to the logical expressions. Each rule translates one or more logical operators into one or more physical operators (algorithms). For instance, a join operator can be implemented as either a merge- or hash-based algorithm, while an interpolation can be implemented by any of variety of curve fitting algorithms. In the process of choosing physical algorithms, the optimizer considers physical issues such as sortedness of data sequences. In the optimization of scientific computations, this choice is complicated by the need to consider numerical accuracy and stability, as well as the diverse implementation techniques for numerical operations.

Using the physical execution plans, a cost calculation determines an estimate of the overall cost to execute each plan. In scientific computations, CPU costs are much more varied than in relational queries, and often dominate

the cost of specific operators. The details of our cost model are discussed in Section 6.3.

It is the role of the database implementor to derive logical transformations, provide implementation rules and define the cost model for the optimizer. As a scientific database system is utilized, new operations and algorithms, with their associated transformation and implementation rules, may need to be added to the system, particularly as the operator sets that we examine are not computationally complete (which would present difficulties in optimization). An extensible optimizer allows this sort of continuing growth; managing this growth will fall to the database implementor.

There are further issues specific to query optimizers such as limiting the search space, detecting common sub-expressions and improving cost estimation, among others. While search efficiency was one of the central concerns in the design and implementation of the Volcano optimizer generator [8], these issues are orthogonal to the optimization of scientific computations, and are not addressed in this paper.

3. Data Types

This research focuses on optimization of computations over bulk types. Three basic bulk types are supported by our current system: sets, time series and spectra, the latter two of which are provided as instances of scientific types. In each case, the bulk type consists of a collection of elements, which we will refer to as records, although they may include structures more complicated than a simple record. In this section, we address the logical view of each bulk type explored in this research.

Sets are viewed similarly to relations in the relational model. Ordering is unimportant on the logical level. The cardinality is the only logical property that can be determined for a set.

Time series are also viewed as sets, but each record is assumed to be tagged with a time value. This model is motivated by the fact that time series are representations of functions, or samples of continuous functions, mapping time to a physical value. Although an implicit sorting by time is utilized by many algorithms, such sorting is not necessary on the logical level.

Attached to a time series are a number of values in addition to the cardinality. Start and stop time, the average and maximum difference between samples, and whether or not the time differences are constant between adjacent records are all important for some operations.

We allow subtypes of time series and other types to be determined by constraints on these logical attributes. This allows the enforcement of logical constraints on data objects through the type maintenance facilities of the optimizer. The most common instance of this in our prototype is the subtyping of time series based upon the enforcement of a constant time difference between consecutive records. This constraint is required by numerous operations, such as filtering and conversion to Fourier space (performed by the Fast Fourier Transform).

Finally, spectra are treated in a manner very similar to time series, but with a frequency attribute attached to each record rather than a time value. The same subtyping facilities are available for spectra.

The logical transformations (discussed in Section 5) ensure type-safety in all transformations, i.e., they ensure that appropriate type conversion operators are always included in the query to insure that operators receive objects of the correct bulk type. The types supported by our current prototype are summarized in Figure 2.

Having introduced the data types modeled within the system, the next section presents the operators that are provided to manipulate these objects.

4. Operators

The operators supported within the system are divided into two groups, logical and physical. Logical operators are those that operate on the logical data types without addressing specific data storage formats or execution algorithms, which are handled by the physical operators. The user's original computation is composed of logical operators, while the final execution plan is expressed to the database in terms of physical operators.

In this section, we present the operators supported by the present system, which are shown in Figure 3. These operators are instances of general scientific operators, and are chosen for their common use in scientific processing and for their illustration of special issues that must be addressed by an optimizer for scientific databases.

For each numeric operator, we discuss the issues raised for optimization by the inclusion of that operator in the algebra. Our prototype relies heavily on the extensibility features of the Volcano optimizer generator, [8] which enables us to explore a variety of alternative logical and physical operators, properties, and optimization rules.

Type	Logical Properties	Physical Properties
Sets	Cardinality	Sort order, Record size
Time series	Start / stop time Average or fixed time deltas	Physical format Sort order, Record size
Spectra	Frequency range Average or fixed frequency deltas	Physical format Sort order, Record size

Figure 2: Logical and Physical Properties

Type	Logical Operators	Physical Operators
Sets	Join	Hash & Merge Join
	Select, Project, Sample	Filter
Time series	Digital Filter, Interpolation, Extrapolation, Regular Sampling	Windowing operator
	Merge	Hash & Merge Join
Spectra	Spectral Filter	Filter
	Merge	Hash & Merge Join
All Types	Math Function	Filter
Conversion	Set→Time series, Set→Spectrum	NOP (requires time or frequency tags)
	Time series→Set, Spectrum→Set	NOP
	Time series→Spectrum	FFT
	Spectrum→Time series	Inverse FFT
Enforcers		Sort

Figure 3: Logical and Physical Operators

4.1. Operators on Sets, Time series and Spectra

As we are extending the relational model to handle new data types and operations, we include the standard relational operators select, project and join. In addition, we add a random sampling operator which can select a random subset of a set, which illustrates the extension of the relational operators to scientific set-based operators. This is the simplest form of extension, as it does not require any new type considerations. Random sampling assigns a fixed probability of inclusion for every record of the set, and may also designate that the cardinality of the output set be fixed. Random sampling acts much as a selection, with the exception that no predicate is enforced on the result set, and the sampling does not depend in any way upon the value of the records in the set.

Three basic time series operators are included in the current prototype. Computation based on nearby records (in time) is a frequent operation in scientific analysis, and digital filtering is a common instance of this type of operation, in which specific frequency ranges are removed or augmented in a time series. This operator recomputes the value of a single record based upon an averaging function applied to nearby records.

An operator for interpolation and extrapolation of time series is included in the prototype as an instance of an operation that can generate new data records in a time series. Regular sampling can be performed by this operator as well. Individual records are added to or removed from a time series by these operations, and thus the logical attributes of the time series can be changed. The actual interpolation formula is generally unimportant to the optimization of the logical computation, although it may influence the physical algorithm cost.

Finally, an operator for merging two time series is included. This is essentially a join by time, though it is necessary that the two time series have corresponding time tags. Typically this is ensured by having identical start and end times, and fixed and equal time deltas between records.

Two operations are provided for manipulating spectra. First, a spectral filter (a frequency filter performed in spectral space) may be applied to a spectrum. This corresponds to the digital filter available for time series, and is illustrative of the many operations which can be performed in either time space or Fourier-space. In many cases, one or the other will be simpler or less expensive to apply, and one may be more accurate than the other. Second, merging of spectra is supported by a merge operation similar to that for time series.

4.2. Other Logical Operators

There are some operations which can be applied to all data types. The primary such operation considered here is a simple math function application, which is included to support many common scientific operations. Through this operation, each output record is generated by a mathematical expression from an input record. Scientific operations such as correlation, convolution and deconvolution of spectra are mathematical functions applied to the result of merging two bulk types. Other examples are relational projection without duplicate elimination, as well as the transformation of a time series consisting of polar coordinates (latitude, longitude, altitude) into an equivalent time series consisting of Euclidean coordinates (x, y, z).

Operators are also used to convert data objects from one type to another, and provide the basis for the type maintenance facilities in our prototype. In some cases, these operators do not require any physical data manipulation. For example, converting a time series or a spectrum to or from a set (with the appropriate record fields for time or frequency) requires no computation.

When converting between time series and spectra, a Fourier transform or its inverse must be applied. As this is generally an expensive operation to perform, the decision on when to move between normal and Fourier-space is important for optimization, illustrating the importance of the type system in the optimization process.

4.3. Physical Operators

Physical operators in the Volcano system are implemented as iterators [6]. Volcano iterators already exist to implement relational operations, as well as sampling, merging, sorting and mathematical function

application. Some additional scientific operations were developed as part of this research.

A new operator has been added to pass a window over a sorted data set. This provides the implementation of algorithms for interpolation and extrapolation, as well as digital filtering. Aspects such as the window size and the function to apply can be set by the user, allowing the operator to support most algorithms requiring a window over a data sequence.

A second operator considered within the system is the Fast Fourier Transform (FFT). We modeled FFTs in two steps which are considered separately by the database. First, a "bit-reverse sort" must be performed, in which each element in the input sequence is re-ordered to a new position in the output sequence, which is the bit-reverse of the original position index. This is normally an O(N) algorithm when performed in memory, but the page access pattern causes poor execution when the data set does not fit into memory, resulting eventually in $O(N \times \log(N))$ performance, similar to other sorting techniques.

The actual Fourier transform, which involves combining each element with every other element in a tree-like pattern, can be performed in $O(N \times \log(N))$ time, and can be adapted to the iterator approach, implemented as a series of $\log_2(N)$ iterators each processing a single level in the FFT processing tree. Each of these operators performs a scan and reordering operation on its input, in addition to the FFT computation.

The next section present the rules used to manipulate expressions of logical operators.

5. Logical Transformations

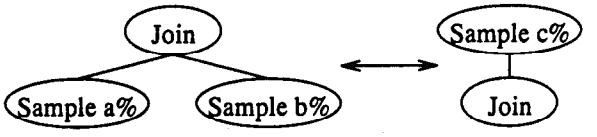
Logical transformations are used to reorganize a computation consisting of operators, such as the application of join associativity in relational systems. Identifying logical transformations for scientific operators is vital to the application of optimization. Of particular importance is identifying transformations with expensive operators, primarily joins and Fourier Transforms.

The central issue in identifying logical transformations is equivalence. This is straightforward in relational transformations, where a transformation results in a logical expression that generates an identical output relation. Due to the effects of numerical accuracy and stability, the results of transformations which include numerical operators occasionally do not produce exactly identical results, even though the transformation is considered valid by the scientific user. This issue can influence which transformations can be used on a given expression. In many cases, some of which are illustrated in the following subsections, the transformations do not change the output of an expression. Thus significant optimizations can be performed without worrying about numerical accuracy and stability.

Nonetheless, many advantageous transformations are available in which numerical accuracy and stability issues are important. This area is open to further research. Here, we present some possible considerations when addressing this issue. First, there are a number of transformations in specific scientific domains that are considered generally valid. For the field of space science, in which we have done most of our examination, there are many filtering operations which can be performed in either time- or Fourier-space. Second, there are transformations which may require input from the user, or notification to the user of their use. In these cases allowing the user to disable or enable certain transformations to control the accuracy of the results of the equation would be desirable. Third, it may be possible for the database optimizer to consider numerical accuracy directly, insuring that the final accuracy does not exceed some limit, or even making a trade-off between accuracy and execution time by factoring accuracy into the cost formulas. These possibilities will be explored through future research.

5.1. Set Operator Transformations

Within our system the standard relational transformations apply. Transformations are also available for the random sampling operator. For sampling which is purely probabilistic (i.e. which makes no assurances about the output cardinality), transformations are similar to those for a relational selection, although attention must be paid to sample-join transformations. Here, a sample may be interchanged with a join only when there is statistical confidence that the selection or rejection of a given record will not skew the join results beyond an accuracy acceptable to the user. When allowed, a sample moved below a join can be split almost arbitrarily between the two join inputs, provided that the product of the two probabilities of the resulting samples equals that of the original sampling operator. This is illustrated in Figure 4. When the sampling must insure a fixed output cardinality (i.e. must select exactly N out of all of the records), the available transformations are limited to those in which the second operator does not modify the output cardinality. For example, in scientific computations it is common to merge two data sets knowing that every record in each input set will match exactly one record in the other set. In this case, a sample may always be performed before or



where a% × b% = c%

Figure 4: Probabilistic Sampling / Join Transformation

after the merge, although performing it before the merge causes the merge to lose that property.

The similarities between sampling and selection transformations show that many transformations available to scientific operators mirror those of existing operators.

5.2. Math Operator Transformations

Math operators, which apply mathematical functions to records in a bulk type, can be transformed with one another using the standard rules for manipulating expressions. In this way they act like relational selections, except that the expressions involved are not limited to Boolean expressions. Similar techniques for splitting off subexpressions and combining multiple expressions can be applied. For such an operator it is useful to maintain those attributes that are used by the operator, and those that are updated or added by the operator. The operator can then be interchanged with any other operator with which it has no input or output conflicts.

The math operator transformations illustrate that many scientific transformations can be identified through an analysis of which attributes an operator reads or writes.

5.3. Time series and Spectra Operator Transformations

Both digital filtering and interpolation operations illustrate transformation restrictions unique to scientific operators. In these cases, the restrictions arise from the window of nearby records used by these operators when determining a single output record. This limits transformations to those that do not add or remove records, or modify the attributes used by these operators, unless the operator can account for this change.

In the case of digital filtering, this prevents most transformations with other operators. Filters can be transformed before or after merge operations when it is known that the merge will be exactly one-to-one, with no unmatched records, and the filter does not use or modify the merge field. Also, filters can often be performed in either time or frequency domains, which we model by allowing a time-based filter followed by a conversion to a spectrum to be replaced by a conversion to a spectrum followed by a spectral filter, as shown in Figure 5.

For interpolations, an additional option is available. As interpolations can account for non-existing values in records, an interpolation occurring before a one-to-one merge can be moved after the merge, provided that the merge or join operator is replaced with the appropriate outer join operation. This is illustrated in Figure 6.

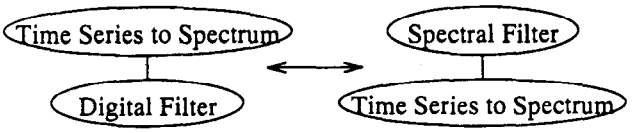


Figure 5: Time / Spectral Space Transformations

5.4. Extensibility of Transformations

In general, when dealing with numeric operators in this framework, it is possible to provide guidance to the scientific database implementor for identifying logical transformations, but it is difficult to ensure that all possible transformations have been identified. This is a consequence of the differing nature of equivalence in scientific computations, and the differing requirements of specific applications in terms of numeric accuracy. As a result, the ability to extend the set of available transformations is important in scientific databases, leading to the need for an extensible optimizer such as the Volcano optimizer generator [8].

6. Implementation Rules

From logical expressions generated through the transformation rules, implementations with physical operators are selected. The cost model is applied to the resulting plans to select an optimal execution strategy.

To convert a logical expression to an execution plan, implementation rules are applied. An implementation rule defines an execution strategy for a sub-expression of one or more logical operators. For instance, in relational systems implementation rules are used to indicate that a join operator can be implemented through either a hash-based or a sort-based algorithm.

Scientific database optimization requires special consideration of implementation rules, as circumstances arise which differ significantly from those in pure relational systems.

6.1. Example Implementation Rules

As one example implemented in the current system, a windowing operator which can pass a window over a sequence of data while performing some operation on the data can be used to implement both interpolation and

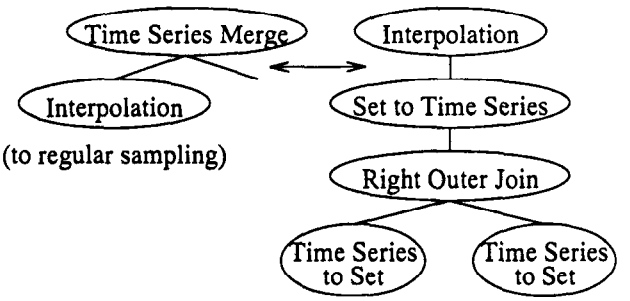


Figure 6: Interpolation / Join Transformation

extrapolation operations as well as digital filtering operations. This situation, where different data manipulations make use of similar algorithms, is not uncommon in scientific processing. If the optimizer is extensible, it is straightforward to implement new logical operators using existing physical algorithms.

A second example is the Fast Fourier Transform (FFT), which is a direct implementation of a type conversion between time series and spectra. There is a well-known technique by which two sequences of the same size which are known to be real (i.e., do not have imaginary components) can be packed together into a single sequence of complex numbers before being processed by the FFT, resulting in a factor of two speed improvement (two FFTs vs. one FFT). This situation is very common, as time series are generally of real data, and correlations require converting two or more time series into spectra simultaneously. The use of this technique is illustrated in the second example in Section 7.2. The FFT algorithm presents an instance of a CPU intensive operator. Such operators are common in scientific applications, and influence the development of the cost model chosen by the database implementor.

6.2. Cost Model

Once the implementation rules have been applied to generate an execution plan, the cost of that plan must be estimated to allow comparison of alternatives. To determine the cost of an operation, the optimizer assumes that, given the properties of its inputs such as cardinality, sortedness, etc., the cost of the operation can be estimated using cost functions supplied by the database implementor. These cost functions may make use of database statistics for selectivity estimation and/or cost computations. It is worth noting that selectivity estimation is often perfect in scientific computations, as a selection of a range of time tags in a time series generates a calculable result size, and many algebraic operations do not change the result size, or else modify it in predictable ways.

It is our belief that the high CPU costs of many scientific operations leads to the need for CPU cost consideration during optimization. Therefore, in our prototype, we have chosen to use time to completion as the basis for optimization. I/O and CPU costs are maintained separately, and we assume that we can utilize asynchronous I/O. The greater of I/O and CPU costs for the computation is used as the time to completion.

Having presented the model used in our research, the next sections give two example computations processed by the existing optimizer, illustrating the applicability of the approach presented in this paper.

7. Examples

In this section we provide two example applications of the prototype optimizer to scientific computations. These examples are presented to illustrate the applicability of this research to real-world scientific computations. In particular, the second example is derived from a computation used at the Space Grant College at the University of Colorado at Boulder, in the analysis of satellite data.

7.1. Example 1

This example illustrates the application of transformation rules involving scientific operators mixed with set-oriented relational operators. The query requires matching two separate sets containing experimental measurements of pressure and temperature values. These values are matched by time tag, then a calculation is applied to the result. The calculation requires a data point every second. The pressure data has measurements every second, but the temperature data has 10-second intervals between values, and will need to be interpolated. This query is shown in Figure 7.

The optimization on this query is performed twice. During the first pass the final output data is requested sorted by time. A second optimization is made with no sorting requirements. The results of the first optimization are shown in Figure 8. In this optimization, the requirement for sorting results in the selection of a merge-join based join algorithm, with the interpolation applied after the join. The same query, when optimized without sorting requirements, results in plan shown in Figure 9. Here, the optimizer has chosen to place the interpolation operator before the join. This allows the database to sort only the smaller (temperature) data set to perform the interpolation. By avoiding the sort for the larger (pressure) data set, the estimated execution time for this query is reduced to one-third of the previous plan's

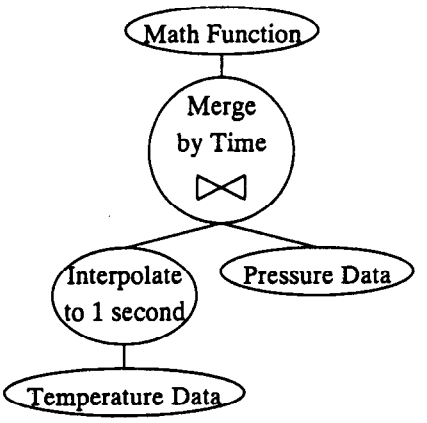


Figure 7: Logical Query for Example 1

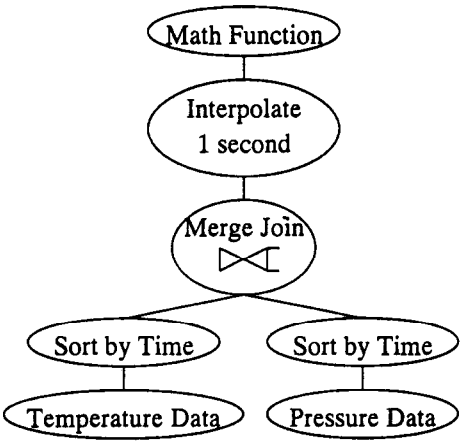


Figure 8: First Execution Plan for Example 1

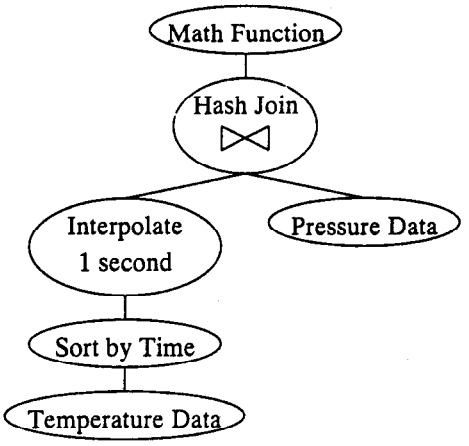


Figure 9: Second Execution Plan for Example 1

estimated execution time.

This example illustrates the benefit of performing scientific operations within the database query execution system, and of providing the capability to optimize those operations in the query optimizer.

7.2. Example 2

This example demonstrates the application of our prototype optimizer to a complex scientific computation, with a number of data preparation and cleaning steps in both time- and Fourier-space. The computation compares, in Fourier-space, high-energy ultraviolet light from the Sun to ozone concentrations in the atmosphere.

The initial computation is shown in Figure 10. It consists of the merging of two pre-processed data sets, one containing the high-energy ultraviolet (EUV) data and the other containing the ozone (O_3) concentrations. The pre-processing is similar for both inputs.

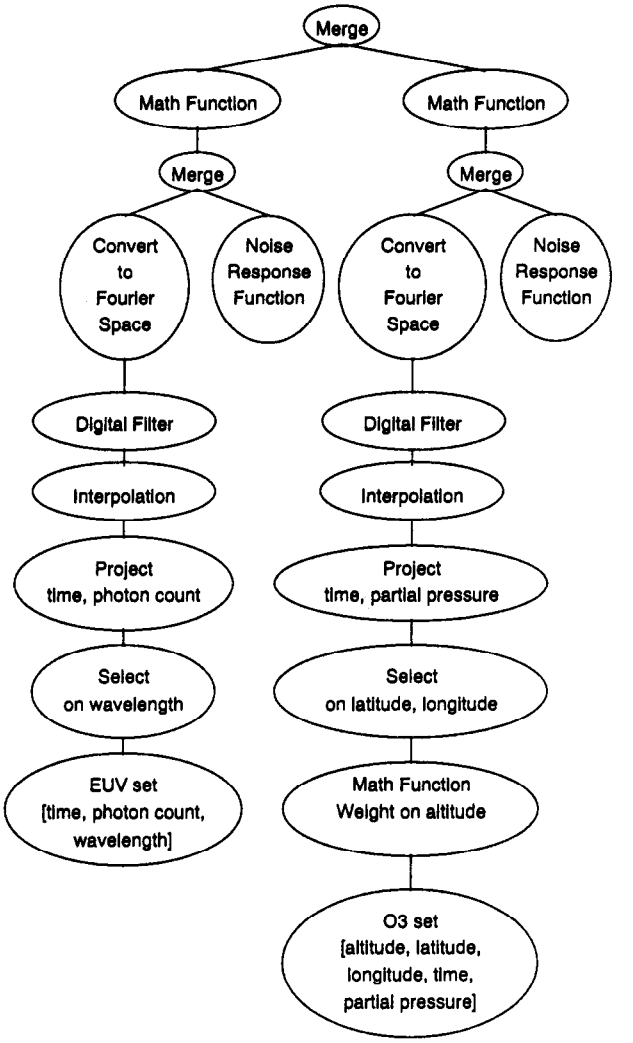


Figure 10: Logical Computation for Example 2

The EUV pre-processing consists of selecting by the measurement wavelength, interpolating the data to a regular sampling frequency, applying a filter function, and convoluting the data with a response function to remove noise sources. This last step is performed in Fourier-space. Similarly, the O_3 processing begins by weighting the data to account for measurement altitudes, selecting the relevant longitude and latitude, interpolating to a regular sampling frequency, applying the filter function, and convoluting the data with the noise response function.

When optimized, the computation is re-written as shown in Figure 11. Pre-processing steps which were identical for both data sets in the initial computation are brought together, and some reordering is performed.

The implementation of the resulting computation realizes a speed improvement greater than a factor of two under our cost model, when compared to the direct

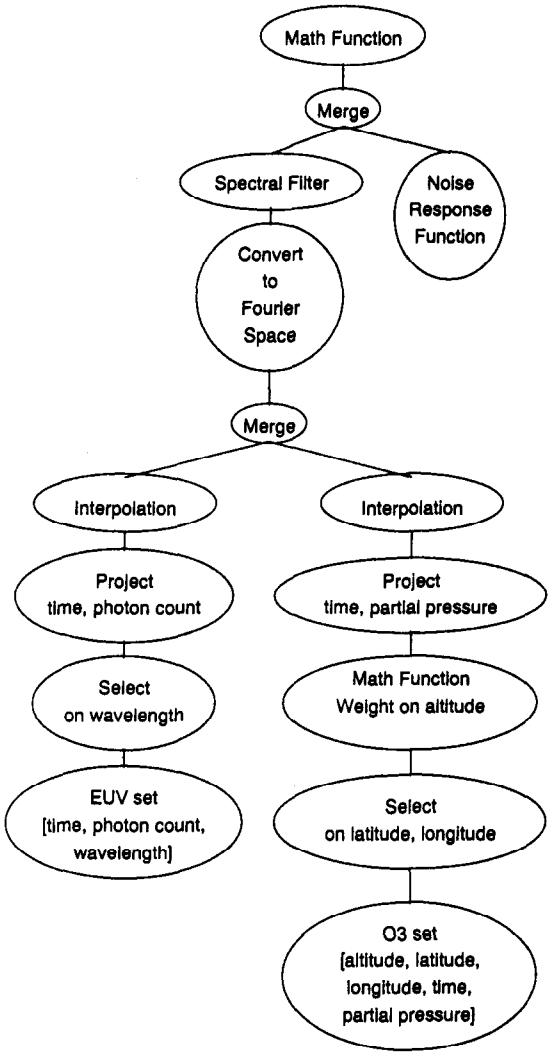


Figure 11: Optimized Computations for Example 2

implementation of the original computation without using any algebraic transformation rules. This improvement arises from three sources: the use of a two-part Fast Fourier Transform algorithm, presented in Section 6.2, the performance of selections at earlier stages of the computation and the reduction of data copying.

By re-writing the computation to perform a single conversion to Fourier-space, the computation can benefit from the use of the two-part FFT. This almost doubles the speed over a pair of normal FFTs (minus some overhead). This change dominates the performance improvement of the computation due to the high cost of FFTs in comparison to other operations.

As in relational optimization, it is desirable to perform selections as early as possible in a computation to reduce data volumes. In this example a performance improvement is realized by pushing the selection by latitude and longitude before the weighting by altitude.

Finally, the reduction in the number of merge operations from 3 to 2 results in less copying of data, and thus better performance.

This example illustrates the applicability of algebraic query optimization to real scientific computations, and shows that significant performance improvements can result from optimization. This examples was taken from an analysis application at CU's Space Grant College.

8. Related Work

A number of researchers have considered the needs of database support for scientific applications [5]. Much of the previous research has focused on identifying the unique requirements of scientific databases. Shoshani, Olken and Wong have enumerated the different kinds of data used in scientific systems [25]. Time series based experimentation data is identified as one of the primary special types of data required within scientific databases. Operators for manipulating the data are not addressed in the paper, though later work by the same authors [26] addresses database operations specifically. Operations for performing sampling, nearest neighbor search, estimation and interpolation, transposition, aggregation and relational operations, integrated into one system, are identified as being of particular importance. The need to support more complex data types, particularly temporal data and time series, is discussed, and query optimization of scientific database operators is mentioned but not examined further. More recent efforts to identify what database systems must provide to meet the needs of the scientific community can be found in two recent survey articles [5, 28]. Among the needs identified are effective use of parallelism and distribution, database extensibility, management of large amounts of data, special data structures and storage formats, and integration of analysis operations into database managements systems. All of this work has focussed primarily upon identifying the needs of scientific database systems, rather than exploring specific solutions to meeting those needs.

Many researchers have investigated data modeling and storage structures for scientific databases. Kim has investigated the application of object-oriented database technology, with the resulting support for complex data types, to scientific storage issues [12]. Kim essentially concludes that, while an object-oriented system provides support for many of the needs of scientific and statistical systems, such as multi-dimensional bulk types, many important scientific database issues are orthogonal to data model issues. Thus, while an object-oriented data model and interface language might be well suited to scientific database management, the issue of user interface languages and models is beyond the scope of this paper.

Special issues for access to and storage of scientific data have been considered [18, 24]. Much research has

been focussed on the special needs of temporal data, which constitutes much of the data in scientific databases [2, 9, 10, 19-23, 27], though this research has dealt with histories, joining historical data and logical reasoning about time in databases, rather than time series data types and optimization technology.

There has been less research on scientific database processing issues. Many similarities exist between scientific operations and statistical operations, which have been explored more extensively. Olken and Rotem have researched operators for random sampling [14-17]. Much of this research has addressed the performance of random sampling from various storage structures such as B+ trees and hash indices. This research illustrates the kind of physical algorithm for non-relational computations that will be required in scientific database systems, but does not address algebraic optimization issues for computations including such operators.

Band joins, are important in scientific databases. These are joins performed with a join argument of the form $R.A + c \leq S.B \leq R.A - d$, where R and S are the relations to be joined, A and B are attributes of the respective relations and c and d are the tolerances that define the width of the band over which the join is to be performed. Algorithms for efficient execution of such joins, using both sort-based and hash-based methods, have been considered by DeWitt, Naughton and Schneider in [4]. This provides another example of a database operator useful in scientific processing. Again, optimization issues are not addressed within this research, as the focus is upon the implementation of a specific database operator.

Neugebauer examined the inclusion of interpolation operations in a relational database system, including some optimizations of the interpolation procedures developed in the research [13]. The problem addressed was the inclusion of interpolation processing, similar to the interpolation operator in our system but potentially in multiple dimensions, into a relational database system. The solution was to add special procedures to a standard relational database system, where the interpolation procedures read in data from the source tables and produce a relation with the correct interpolated data. SQL syntax to incorporate these procedures was also supplied. The results, while accomplishing the objective of providing interpolation capability to a database system, illustrate the difficulties encountered when forcing scientific data processing into a relational model, with no support for sequence data types or non-set-oriented operators.

Other systems have explored extensibility in optimization technology, including EXODUS [3] and Starburst [11]. Extensible optimization technology is a necessary foundation for the research in this paper. For a more complete discussion of this technology and further references, see [1, 8].

9. Summary and Conclusions

This paper has explored the integration of traditional database pattern matching operators and numeric scientific operators. We have shown that a mixed algebra and type model can be used to perform algebraic specification and optimization of scientific computations. The model developed in this research supports operators on multiple bulk types, such as sets, time series and spectra.

We conclude from our experimental results that there is a clear, realizable benefit to be achieved from removing the barrier between the database system and scientific computations over the database. Removing this barrier entails incorporating scientific operations and data types into the query processing framework of the database. Thus, automatic optimization of integrated algebras is a crucial step in supporting scientific computations over database systems.

As part of our work on database support for computational models of turbulence, we are developing a more complete system based on these concepts and built on the Volcano extensible database system. This work is being performed in conjunction with the Space Grant College at the University of Colorado, where it will be applied to the analysis of atmospheric and solar data from satellite and other space science experiments.

In addition to sets, time series and spectra, scientific processing often requires the analysis of large, multi-dimensional arrays. Integrating support for arrays, as well as operations on them, is an important extension of this research which we are currently investigating.

Acknowledgements

We would like to thank Elaine Hansen and Tony Colaprete, of Space Grant College, for their help in applying this research to real-world applications.

This paper is based on research partially supported by the National Science Foundation with grants IRI-8912618, IRI-9116547, IRI-9119446, and ASC-9217394, ARPA with contract DAAB 07-91-C-Q518, Texas Instruments, and Digital Equipment Corp.

Appendix A

This appendix presents the operators, transformation rules and implementation rules in our prototype scientific optimizer. It also provides example code from the model input file used to implement the system under the Volcano optimizer generator.

Figure 3, presented earlier, lists the operators used in our prototype. Operators are specified by type, name and number of inputs. For instance,

```
%operator JOIN 2
%algorithm HASH_JOIN 2
```

specifies logical and physical join operators.

Figure 12 lists the transformation rules applicable to set operators, primarily the standard relational transformations. For some transformation rules, there are conditions which must be met before the rule can be applied. The join / merge interaction requires the inclusion of type conversion operations. Likewise, interactions between join and sample operators require that the split be statistically valid. Finally, some interactions require that attributes remain valid, such as in the case of a math operator and a projection operator, where the projection must not eliminate attributes used by the math operator.

As an example transformation rule, the join associativity rule is coded as follows:

```
%trans_rule (JOIN ?op_arg1
      ((JOIN ?op_arg2 (?1 ?2)) ?3))
  -> (JOIN ?op_arg3
      (?1 (JOIN ?op_arg4 (?2 ?3))))
%cond_code {{
  /* Check the validity of this transformation */
}}
%appl_code {{
  /* Set up the ?op_arg3 and ?op_arg4 arguments. */
}}
```

Figure 13 lists the transformation rules applicable to time series and spectrum operators, as well as the any bulk type and type conversion operators. Once again, some interactions require type conversion operators, such as the equivalence between digital and spectral filters. Also, as noted earlier, interpolation / join interactions can convert the join operation into an outer join. Finally, conversions between sets and other types require that the set have appropriate time or frequency attributes.

Operators	Rule	Description
Join	CM	Join inputs reverse
Join	AS	Consecutive joins associate
Join, Merge	EQ	Joins change to and from merges
Join, Project	NI	Projections push through joins
Join, Select	NI	Selections push through joins
Join, Sample	NI	Samples push through joins
Join, Math	NI	Math operators push through joins
Project	EL	Projections become the last projection
Project, Select	NI	Projections push through selections
Project, Sample	NI	Projections push through samples
Project, Math	NI	Projections push through math operators
Select, Sample	NI	Selections push through samples
Select, Math	NI	Selections push through math operators
Sample	CB	Two samples combine
Sample, Math	NI	Samples push through math operators

Figure 12: Set Operator Logical Transformation Rules

Operators	Rule	Description
Interpolate	EL	Interpolations become the last interpolation
Digital Filter	CB	Two digital filters combine
Digital Filter, Spectral Filter	EQ	Digital filters change to and from spectral filters
Spectral Filter	CB	Two spectral filters combine
Merge	CM	Merge inputs reverse
Merge	AS	Consecutive merges associate
Merge	EL	Identical merges become one merge
Merge, Interpolate	IN	Interpolations push through merges
Merge, Digital Filter	NI	Digital filters push through merges
Merge, Math	NI	Math operators push through merges
Merge, Spectral Filter	NI	Spectral filters push through merges
Math	CB	Two math operators combine
Conversions	CB	Multiple conversions combine

Figure 13: Time Series & Spectrum Logical Transformation Rules

On these two tables, rule names are abbreviated as follows: **AS** Associativity, **CB** Combination, **CM** Commutativity, **EL** Elimination, **EQ** Equivalence, **IN** Interchange, **NI** Non-interference.

Figure 14 lists the implementations rules used to implement the logical operators, along with the conditions under which the implementation rule can be applied. The code for the hash join implementation rule is given below:

```
%impl_rule (JOIN ?op_arg1 (?1 ?2))
  -> (HASH_JOIN ?al_arg1 (?1 ?2))
```

Condition code can also be included on implementation rules. In our prototype, it is only required for the FFT implementation, in which the input sequence must have a fixed delta between records.

References

[1] J. A. Blakeley, W. J. McKenna and G. Graefe, "Experiences Building the Open OODB Query Optimizer", *Proc. ACM SIGMOD Conf.*, Washington, DC, May 1993, 287.

[2] A. Bolour, T. L. Anderson, L. J. Dekeyser and H. K. T. Wong, "The Role of Time in Information Processing: A Survey", *ACM SIGMOD Record 12*, 3 (April 1982).

Logical Operators	Physical Algorithms
Join, Merge	Hash Join, Merge Join
Project, Select, Sample Spectral Filter, Math	Filter
Interpolate, Digital Filter	Window
Set→Time series, Set→Spectrum, Time series→Set, Spectrum→Set	NOP
Time Series→Spectrum, Spectrum→Time series	FFT
Get (Set, Time series, Spectrum)	File Scan, Btree Scan

Figure 14: Physical Implementation Rules

[3] M. J. Carey, D. J. DeWitt, G. Graefe, D. M. Haight, J. E. Richardson, D. T. Schuh, E. J. Shekita and S. Vandenberg, "The EXODUS Extensible DBMS Project: An Overview", in *Readings on Object-Oriented Database Sys.*, D. Maier and S. Zdonik (editor), Morgan Kaufman, San Mateo, CA, 1990.

[4] D. J. DeWitt, J. E. Naughton and D. A. Schneider, "An Evaluation of Non-Equijoin Algorithms", *Proc. Int'l. Conf. on Very Large Data Bases*, Barcelona, Spain, September 1991, 443.

[5] J. C. French, A. K. Jones and J. L. Pfaltz, "Scientific Data Management", *ACM SIGMOD Record, Special Issue on Directions for Future Database Research and Development 19*, 4 (December 1990), 32.

[6] G. Graefe, "Volcano, An Extensible and Parallel Dataflow Query Processing System", *to appear in IEEE Trans. on Knowledge and Data Eng.*, 1993.

[7] G. Graefe, R. L. Cole, D. L. Davison, W. J. McKenna and R. H. Wolniewicz, "Extensible Query Optimization and Parallel Execution in Volcano", in *Query Processing for Advanced Database Applications*, J. C. Freytag, G. Vossen and D. Maier (editor), Morgan-Kaufman, San Mateo, CA, 1993.

[8] G. Graefe and W. J. McKenna, "The Volcano Optimizer Generator: Extensibility and Efficient Search", *Proc. IEEE Conf. on Data Eng.*, Vienna, Austria, April 1993, 209.

[9] H. Gunadhi and A. Segev, "A Framework for Query Optimization in Temporal Databases", *Proc. Fifth Int'l. Conf. on Statistical and Scientific Database Management*, April 1990.

[10] H. Gunadhi and A. Segev, "Query Processing Algorithms for Temporal Intersection Joins", *Proc. IEEE Conf. on Data Eng.*, Kobe, Japan, April 1991, 336.

[11] L. Haas, J. C. Freytag, G. Lohman and H. Pirahesh, "Extensible Query Processing in Starburst", *Proc. ACM SIGMOD Conf.*, Portland, OR, May-June 1989, 377.

[12] W. Kim, "Object-Oriented Approach to Managing Statistical and Scientific Databases", *Proc. Fifth Int'l. Conf. on Statistical and Scientific Database Management*, April 1990.

[13] L. Neugebauer, "Optimization and Evaluation of Database Queries Including Embedded Interpolation Procedures", *Proc. ACM SIGMOD Conf.*, Denver, CO, May 1991, 118.

[14] F. Olken and D. Rotem, "Simple Random Sampling from Relational Databases", *Proc. Int'l. Conf. on Very Large Data Bases*, Kyoto, Japan, August 1986, 160.

[15] F. Olken and D. Rotem, "Random Sampling from B+ Trees", *Proc. Int'l. Conf. on Very Large Data Bases*, Amsterdam, The Netherlands, August 1989, 269.

[16] F. Olken and D. Rotem, "Random Sampling from Database Files: A Survey", *Proc. Fifth Int'l. Conf. on Statistical and Scientific Database Management*, April 1990.

[17] F. Olken, D. Rotem and P. Xu, "Random Sampling from Hash Files", *Proc. ACM SIGMOD Conf.*, Atlantic City, NJ, May 1990, 375.

[18] R. K. Rew and G. P. Davis, "NetCDF: An Interface for Scientific Data Access", *Computer Graphics and Applications*, July 1990, 76.

[19] D. Rotem and A. Segev, "Physical Organization of Temporal Data", *Proc. IEEE Conf. on Data Eng.*, Los Angeles, CA, February 1987, 547.

[20] A. Segev and A. Shoshani, "Logical Modeling of Temporal Data", *Proc. ACM SIGMOD Conf.*, San Francisco, CA, May 1987, 454.

[21] A. Segev and A. Shoshani, "The Representation of a Temporal Data Model in the Relational Environment", *Proc. Fourth Int'l. Working Conf. on Statistical and Scientific Database Management*, June 1988.

[22] A. Segev and H. Gunadhi, "Event-Join Optimization in Temporal Relational Databases", *Proc. Int'l. Conf. on Very Large Data Bases*, Amsterdam, The Netherlands, August 1989, 205.

[23] R. Shodgrass and I. Ahn, "A Taxonomy of Time in Databases", *Proc. ACM SIGMOD Conf.*, Austin, TX, May 1985, 236.

[24] A. Shoshani, "Statistical Databases: Characteristics, Problems, and Some Solutions", *Proc. Int'l. Conf. on Very Large Data Bases*, Mexico City, Mexico, September 1982, 208.

[25] A. Shoshani, F. Olken and H. K. T. Wong, "Characteristics of Scientific Databases", *Proc. Int'l. Conf. on Very Large Data Bases*, Singapore, August 1984, 147.

[26] A. Shoshani and H. K. T. Wong, "Statistical and Scientific Database Issues", *IEEE Trans. on Softw. Eng. 11*, 10 (October 1985).

[27] A. Shoshani and K. Kawagoe, "Temporal Data Management", *Proc. Int'l. Conf. on Very Large Data Bases*, Kyoto, Japan, August 1986, 79.

[28] A. Silberschatz, M. Stonebraker and J. D. Ullman, "Database Systems: Achievements and Opportunities", *ACM SIGMOD Record, Special Issue on Directions for Future Database Research and Development 19*, 4 (December 1990), 6.