CESDIS TR 92-81, March, 1992

# MR-CDF: Managing Multi-Resolution Scientific Data[1]

Kenneth Salem

*Department of Computer Science*
*University of Maryland*
*College Park, Maryland 20742*
*and*
*CESDIS*
*NASA Goddard Space Flight Center, Code 930.5*
*Greenbelt, MD 20771*

**Abstract**

MR-CDF is a system for managing multi-resolution scientific data sets. It is an extension of the popular CDF (Common Data Format) system. MR-CDF provides a simple functional interface to client programs for storage and retrieval of data. Data is stored so that low-resolution versions of the data can be provided quickly. Higher resolutions are also available, but not as quickly. By managing data with MR-CDF, an application can be relieved of the low-level details of data management, and can easily trade data resolution for improved access time.

[1]A shorter version of this paper appeared in the proceedings of the Goddard Conference on Mass Storage Systems and Technologies, September, 1992, pages 101-111. The proceedings are NASA Conference Publication 3198.

# MR-CDF: Managing Multi-Resolution Scientific Data

**Abstract**

MR-CDF is a system for managing multi-resolution scientific data sets. It is an extension of the popular CDF (Common Data Format) system. MR-CDF provides a simple functional interface to client programs for storage and retrieval of data. Data is stored so that low-resolution versions of the data can be provided quickly. Higher resolutions are also available, but not as quickly. By managing data with MR-CDF, an application can be relieved of the low-level details of data management, and can easily trade data resolution for improved access time.

## 1 Introduction

Scientific data management libraries, such as NASA's publicly-distributed Common Data Format (CDF), implement simple data models that are tailored for scientific data. Data managed using these libraries is machine-independent, portable, and self-describing. Access to the data is performed through a set of interface functions that shield the details of storage and retrieval from application programs. The libraries provide a common interface upon which portable, application-specific tools (e.g., classifiers, analysis packages, visualization and browsing tools) can be implemented.

Because scientific data sets are often voluminous, it is desirable to make them available at several different resolutions. Preliminary examination, or *browsing*, of large amounts of data often can be performed efficiently using low-resolution data. Tentative analyses can be performed using intermediate-resolutions, and the final analysis can be performed using the data's full resolution. Lower resolution data is desirable during the preliminary stages because it allows large volumes of data to be considered in a reasonable amount of time.

This paper describes a scientific data management library called MR-CDF (Multi-Resolution Common Data Format) which permits multiple-resolution data sets to be manipulated through a simple, functional interface. Application programs that use MR-CDF see a scientific data model identical to that supported by CDF. When retrieving data, however, they are able to specify a desired resolution level. Applications requiring full-resolution data can obtain it, while those that can use lower resolutions are be able to do so simply and quickly.

MR-CDF uses a *multi-stage* representation for stored multi-resolution data. This is illustrated in Figure 1. A data set that is to be made available at $R$ different resolutions is decom-

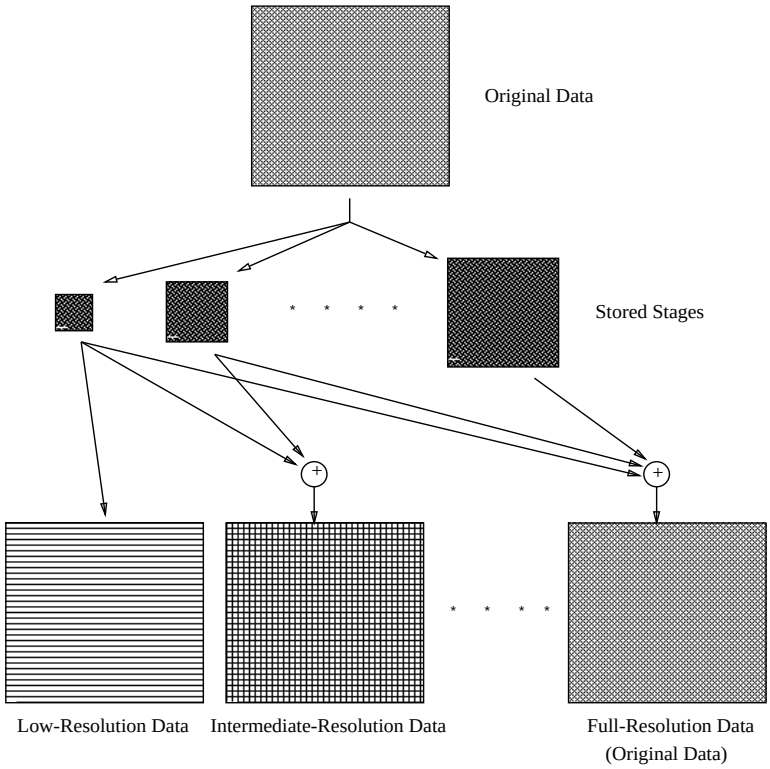



Figure 1: The Multi-Stage Representation Used by MR-CDF

posed into $R$ *stages*, each of which is stored. The decomposition is such that, by retrieving and combining $i$ stages MR-CDF can produce the data at one resolution, and by retrieving $i+1$ stages it can produce the data at a higher resolution. By retrieving and combining *all* of the stages, MR-CDF can produce an exact reconstruction of the data at its original, full resolution. The process of retrieving and combining stages is completely transparent to the application that requested the data, except that lower resolution requests can be satisfied more quickly than others.

There are several difficulties involved in providing an abstract interface for multi-resolution data. The first is the wide variety of techniques that can be used to decompose data into stages for storage. As we shall describe shortly, the decomposition process is essentially an iterative lossy compression of the data. A wide variety of compression techniques are available, and different techniques are well-suited to different types of data. Examples include various region averaging algorithms, vector quantization, and quadtree-like methods [TiMa90, Ti89]. The procedure for properly recombining the stages when data is retrieved depends on which of the many possible compression techniques was originally used to decompose the data. Tying MR-CDF to any particular compression technique would severely limit its applicability. Instead, MR-CDF must be flexible enough to accommodate a wide variety of application-specified techniques.

A second difficulty arises when applications make use of MR-CDF's simple selection facility to retrieve only a portion of the stored data set. Ideally, MR-CDF would perform the selection *before* recombining the stages to minimize the volume of data to be retrieved and recombined. This may or may not be possible, depending on which technique was used to produce the stored stages. Some compression techniques are better suited than others for producing easily-manageable data, at least within the framework of MR-CDF. Although such problems need not limit the functionality of MR-CDF, they may impact its efficiency.

In the remainder of this paper, we describe the design, interface, and implementation of the MR-CDF library. The next section provides an overview of the features MR-CDF and of CDF, on which MR-CDF is based. Sections 3, 4 and 5 describe the relationship between data compression and MR-CDF, and how multi-resolution data is stored into and retrieved from an MR-CDF archive. MR-CDF's selection facility is discussed in Section 6. Finally, Section 7 describes its implementation, which uses CDF's data storage and retrieval facilities.

## 2 What Does MR-CDF Do?

Data management libraries such as CDF do not attempt to provide a solution to the entire scientific data management problem. A CDF archive is designed to hold a set of related, similarly organized scientific data, such as a set of images or a stream of sensor data. The important task of organizing and managing multiple data sets is left to some type of meta-database, such as those described in [RoCa90, ShWa88], and is beyond the scope of both CDF and MR-CDF.

What CDF *does* provide is a simple, abstract programming language interface to scientific data. CDF's capabilities can be summarized as follows.

- CDF provides simple functional interfaces for application programming languages, through which data can be organized, stored, and retrieved.
- CDF implements a data model tailored to scientific applications.
- CDF implements a selection facility which allows applications to specify portions of the database to be retrieved or updated.
- CDF uses a space-efficient internal data representation which is well-suited to many types of scientific data.

Clearly, CDF provides only a subset of the facilties normally associated with database management systems. In particular, it does not provide a high-level language (like SQL) for defining

and manipulating data. However, CDF fits quite naturally as a back-end data manager for many existing scientific data visualization and analysis tools. Futhermore, it is publicly available (with source-code) and is widely used both inside and outside NASA. Although the techniques described in this paper are applicable to other data management systems, we focused on CDF for the reasons above and because it is relatively simple to work with.

The MR-CDF library extends CDF to include support for multi-resolution data sets. It has all of the capabilities of CDF plus the following.

- MR-CDF allows selected data to be retrieved several different levels of resolution. Lower-resolution data can be retrieved more quickly than higher-resolution, allowing applications to trade-off retrieval time for resolution.

- Multi-resolution retrieval in MR-CDF is *progressive*. This means that once a low resolution version of the data has been retrieved, a higher resolution version of the same data can be retrieved in less time than would be required to retrieve the higher resolution data from scratch.

The multi-resolution capability of MR-CDF makes it simple for application programs to select a resolution that is suitable for the task at hand. Progressive retrieval is well-suited to applications such as data browsing. For example, an image browsing program can provide access to many low-resolution images quickly. When an interesting image is found, a progressive retrieval capability allows the browser to provide a higher-resolution version of the interesting image without retrieving the information contained in the low-resolution image a second time.

## 2.1 CDF

The MR-CDF library is based on CDF (Common Data Format) software[Tr90, TrGo90], which is designed to provide a simple, abstract interfact to scientific data. CDF includes interfaces for C and Fortran applications, and is publicly distributed by NASA through the National Space Science Data Center. Since MR-CDF provides a superset of the capabilities of CDF, the following brief description of CDF applies to MR-CDF as well.

Logically, a CDF archive consists of a set of $d$-dimensional records and a set of variables, or parameters. Each variable may have a value at every point in each record's $d$-dimensional data space. The number of dimensions, $d$, and their sizes are specified at the time the CDF archive is created. However, variables added or deleted at any time.

Each variable is typed, and all of a variable's values are of that type. A variety of types are supported, including floating point and integer numbers of various sizes. Array types are also

supported.

When a CDF variable is defined, it is associated with one or more of the dimensions of the data space. A variable's value is only permitted to vary along the dimensions with which it is associated. This mechanism allows CDF to store repeated variable values only once. This can result in tremendous savings of space in scientific data sets, where repeated values are commonly found.

As a simple example, consider a 3-dimensional CDF with four variables representing latitude, longitude, pressure (altitude), and temperature. Suppose that the latitude, longitude, and pressure variables are each associated with a (unique) dimension of the CDF, and that the temperature variable is associated with all three dimensions. Each point in the data space describes the the atmospheric temperature at a particular latitude, longitude and pressure. Such a CDF, with two records, is illustrated in Figure 2.

CDF provides a simple selection facility. When storing or retrieving data from a CDF variable, an application must specify a range along each dimension of the archive as well as the name of the variable. The variable's values within the specified range are then transferred between the CDF archive and an application buffer. The example in Figure 2 shows how an application could retrieve atmospheric temperature data for a specified range of latitude, longitude, and pressure. The selection facility also allows the application to specify a set of records whose data is of interest.

CDF also allows *attribute-value* pairs to be associated with variables, with records, or with the entire archive. One use of attributes is to store meta-data. For example, the temperature variable can be given an attribute called "instrument", whose value would be the name of the instrument that was used to collect the data.

All access to the data in a CDF is performed through a set of interface functions. Functions also exist for creating, retrieving, and storing attribute values, for creating new variables, and for obtaining general information about an archive, such as its dimensionality or the names of the variables it stores.

## 2.2 Multi-Resolution Data

Figure 3(a) shows a plot of some time-series data representing a hypothetical measured quantity "MEAS". Data of this type might be stored as a variable in a one-dimensional CDF archive. For the purposes of this example, suppose that the MEAS variables is of the CDF-defined floating-point type "REAL-4". We will use this example to describe how multi-resolution data in MR-CDF is viewed by application programs.

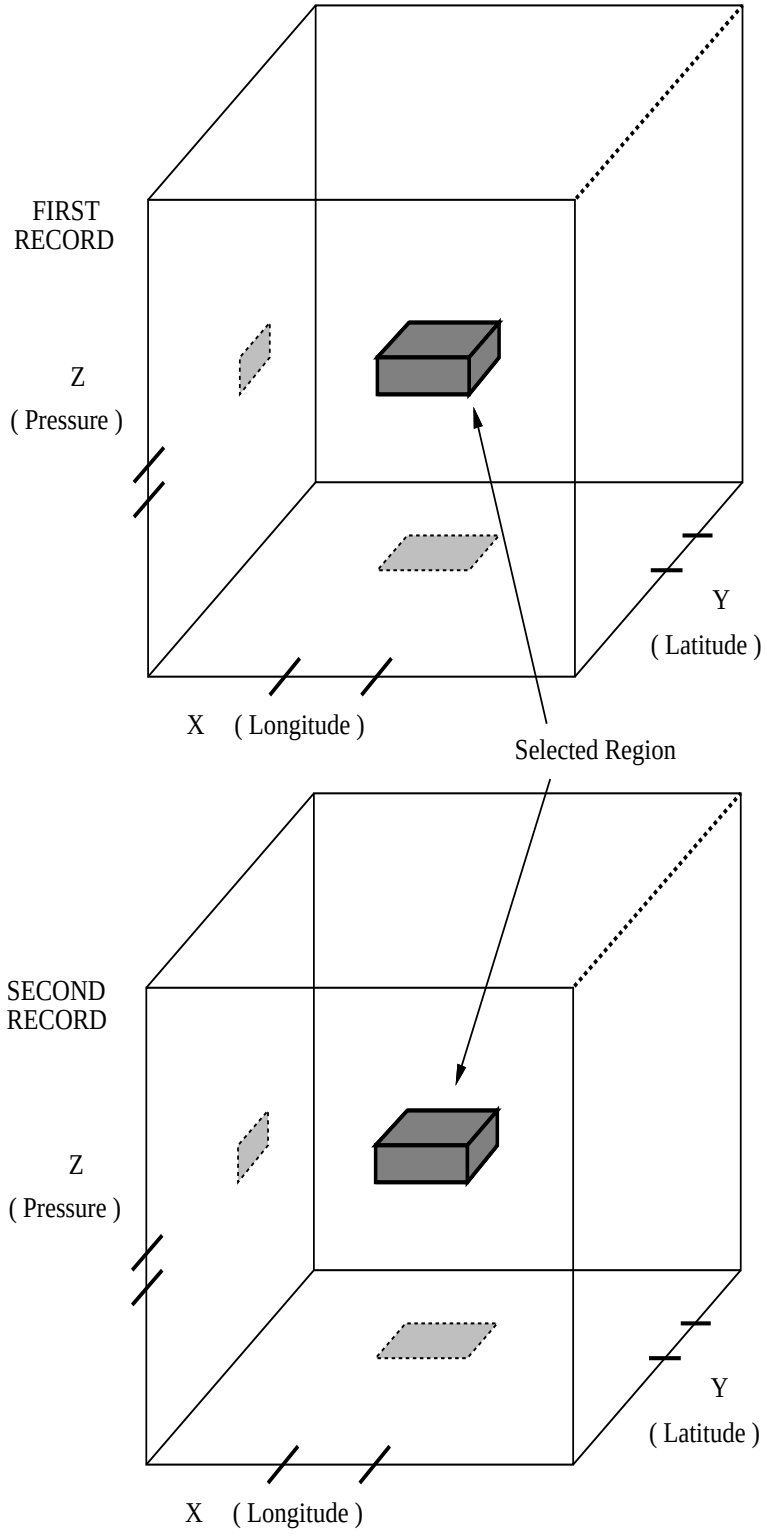
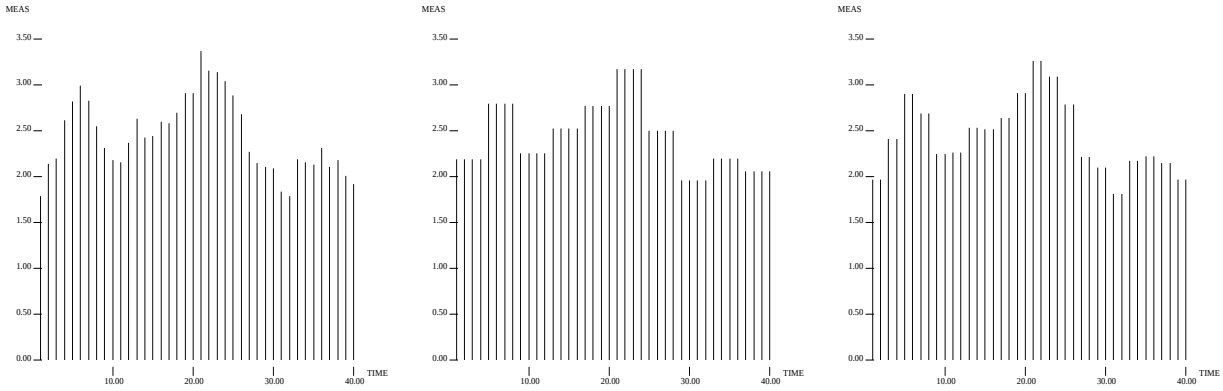



Figure 2: Two Records of A Three Dimensional CDF and a Region Specified by Ranges of X, Y, and Z Coordinates




(a) Full Resolution (b) Low Resolution (c) Medium Resolution

Figure 3: A Simple Time-Series Data Set

MR-CDF adds progressive, multi-resolution retrieval capability to the capabilities of the CDF library. In MR-CDF, one or both of the variables in our example could be stored as *multi-resolution variables*. When a multi-resolution variable is created in an MR-CDF archive, the number of resolutions at which it can be made available is defined. Applications retrieve the values of multi-resolution variables exactly as they would a single-resolution variable, except the desired *resolution level* must be specified as well. A resolution level is specified as an integer between zero and the the number of resolutions defined for that variable, minus one. Smaller numbers represent lower resolutions.

Suppose that "MEAS" is stored as a variable with three possible resolutions. If an application retrieved "MEAS" at resolution zero, it might receive the data plotted in Figure 3(b). Resolution one data might look as plotted in Figure 3(c), while resolution two data would match the full-resolution data in Figure 3(a) exactly.

An important feature of MR-CDF is that the application will receive the same volume and type of data, regardless of the resolution level requested. In our example, the application can expect to receive ten REAL-4 values, regardless of resolution. This greatly simplifies data handling in MR-CDF applications. From the application's perspective, the advantage of lower resolution data is that MR-CDF can provide it more quickly. Although the volume of data passed to the application is independent of the resolution, MR-CDF needs to retrieve less data from its archive to produce the lower resolutions.

The low resolution data in Figure 3(b) were obtained by averaging groups of four values from the original series (Figure 3(a)) and replacing the values in each group by their average. This averaging procedure is a form of *lossy data compression*, since the low resolution series can be represented using a quarter of the values required for the original series. MR-CDF is specifically

designed to manage multi-resolution data that is produced by applying lossy compression to the full-resolution data. Of course, there are many more effective and sophisticated compression techniques than the averaging procedure used in the example. However, we will continue with this example in the following sections because it is simple to describe.

# 3 Data Compression and MR-CDF

Multi-resolution data can be produced from single resolution data by applying lossy compression one or more times. If compression is applied properly, the result is several *stages* of compressed data which can be decompressed and combined to produce the original data set at various resolutions. This process of decompression and recombination is central to MR-CDF.

Many types of lossy data compression have been developed, and different types are well-suited to different types of data. The purpose of MR-CDF is not to define a new compression technique, nor to specify a particular technique that must be used to produce the multi-resolution data. Instead, MR-CDF is designed to be flexible enough to incorporate multi-resolution data produced by a wide variety of compression techniques.

Data compression is not performed by the MR-CDF library. Instead, it is assumed that the compressed data is produced externally and then stored in the MR-CDF archive. MR-CDF performs the decompression and combination of the stored data in response to application requests.

In principle, it would be possible for compression to be implemented within MR-CDF. In an ideal scenario, data would be supplied to MR-CDF in its original full-resolution form, and would be available to applications at several resolutions. In practice, however, compression of the data is often much more time consuming than decompression. Many compression algorithms are best performed on highly parallel machines or with special purpose hardware. For example, compressing data using vector quantization involves vectorizing the input data and comparing each vector against a "codebook" of vectors to find the closest match. Using parallel hardware, the input vector can be compared against all of the codebook entries simultaneously. Decompressing the data involves much less work, since only a simple lookup in the codebook is all that is required to recreate each vector.

Although MR-CDF does not perform data compression, it supports a very general compression model so as to restrict as little as possible the types of compression algorithms whose output can be managed by the library. MR-CDF's decompression procedure is actually a general framework which can be customized to support data produced by a wide class of compression

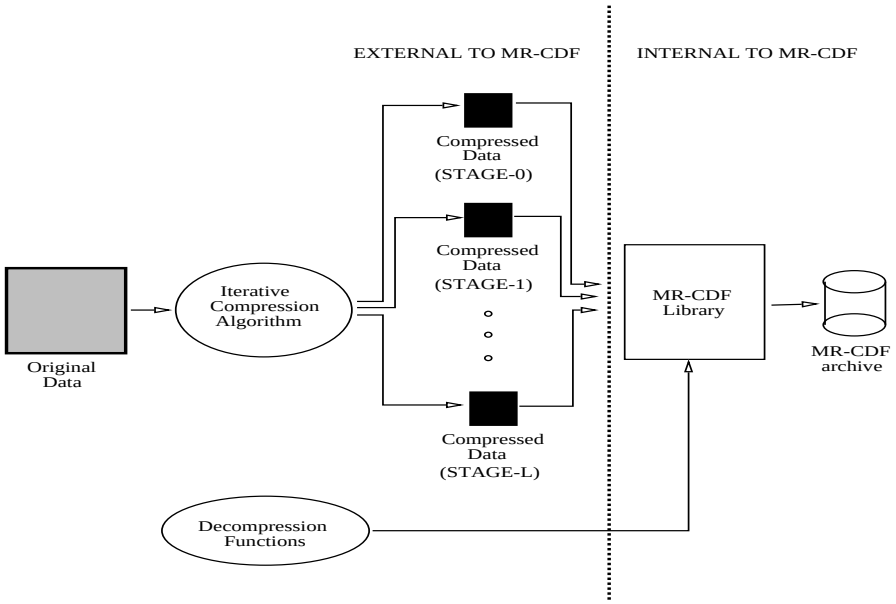



Figure 4: Creating Data for MR-CDF

techniques.

# 4 Producing Data for MR-CDF

Figure 4 illustrates how data suitable for progressive, multi-resolution retrieval is produced and stored in MR-CDF. An iterative compression technique (described below) is applied to the original full-resolution data, resulting in several stages of compressed data. The compressed data are stored in the MR-CDF archive. The stages are such that MR-CDF will be able to recreate the data at resolution level $i$ by retrieving stages 0 through $i-1$ from the archive and then decompressing and recombining them. We will describe the retrieval procedure in more detail shortly.

The iterative compression algorithm shown in the figure actually represents a general class of compression procedures. During each iteration, data is compressed using *some* lossy compression technique, and then decompressed. The difference between the decompressed data and the original is computed. This difference, or error, becomes the data that is compressed during the next iteration.

The iterative compression procedure for computing three stages of compressed data is illustrated in more detail in Figure 5. As illustrated, a lossy compression function $f_i$ is used to produce the stage-$i$ data. Since the compression function is lossy, the decompressed data will not match the original data exactly. The difference between the original data $D_0$ and the decom-

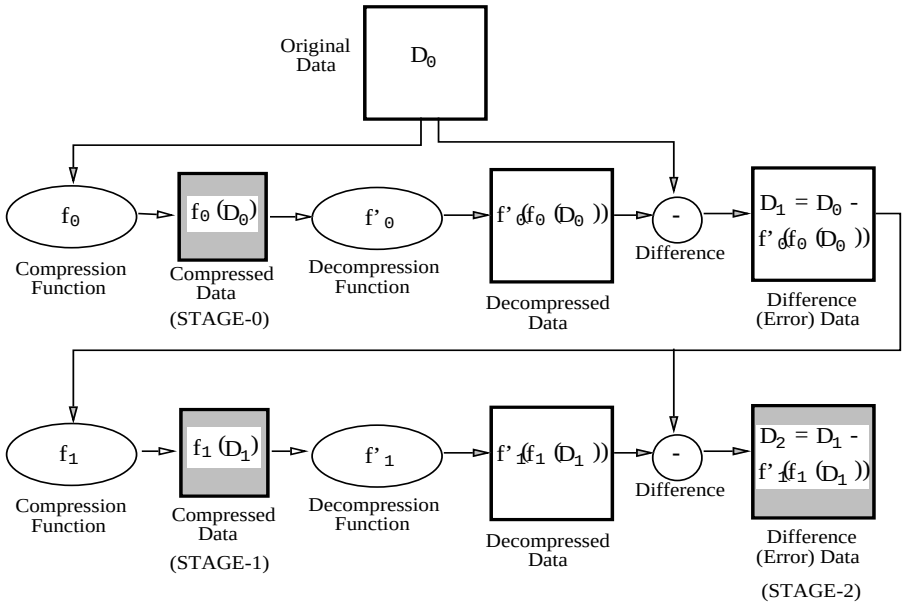



Figure 5: Iterative Compresson Procedure - Three Stages

pressed data $f_i'(f_i(D_i))$ is the error data, which is used as the input to the next decompression stage. In the figure, the shaded boxes represent the compressed data stages which are actually stored in MR-CDF.

The example in Figure 5 may be somewhat misleading since it suggests that the compressed data stages together occupy more space than does the original, full-resolution data set. In practice, this need not be the case. For more realistic compression techniques, such as those described in [TiMa90, Ti89], the stages taken together are about as voluminous as the original data. The compressed stages can be thought of as an alternative representation of the original data which makes multi-resolution retrieval more convenient.

# 5 Retrieving Data from MR-CDF

When an application requests data from MR-CDF, it specifies a desired resolution level. If the iterative compression procedure produced $R$ stages of compressed data, the data will be available at $R$ resolutions. The highest resolution level always corresponds to the original, full resolution of the data.

MR-CDF supplies the data at the specified resolution by retrieving one or more of the compressed data stages from the archive. To retrieve data a resolution $i$, stages 0 through $i$ are retrieved. The retrieved stages are then decompressed and combined to produce the desired data.

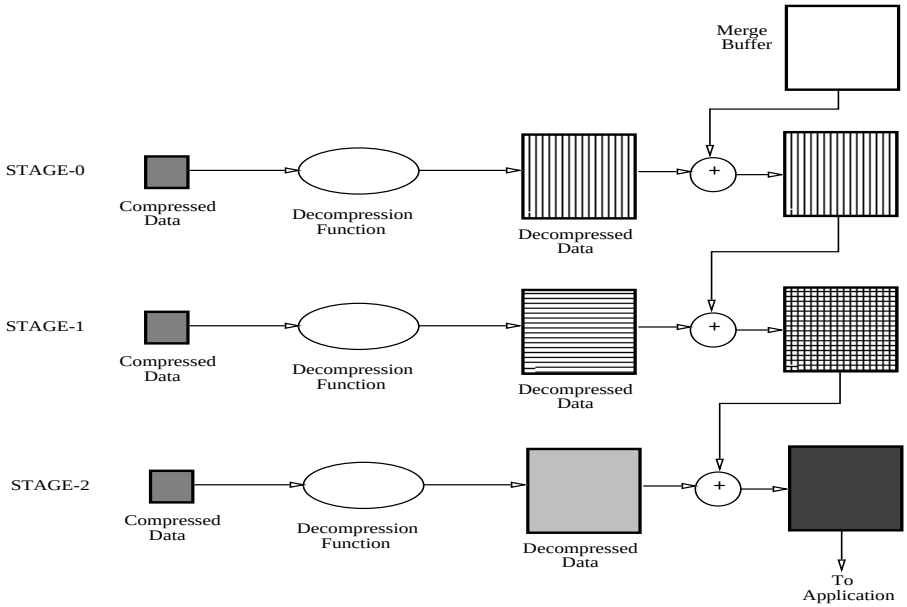



Figure 6: Decompression Procedure - Three Stages

Figure 6 illustrates how MR-CDF would handle a request to retrieve the data compressed as illustrated in Figure 5 at resolution level two. (In this case, resolution level two corresponds to the original, full-resolution data.) Since resolution level two is requested, MR-CDF retrieves the stage-0, stage-1, and stage-2 compressed data. The first two stages are decompressed, and the resulting data is additively merged into a single buffer. In this case, the buffer will contain an exact recreation of the original data $D_0$.

If a lower resolution level is specified, MR-CDF need only retrieve and decompress some of the stages. For example, for resolution level 0, only the stage-0 data is retrieved and decompressed.

## 5.1 Decompression Functions

MR-CDF's retrieval procedure requires that a set of decompression functions be available. Since an arbitrary compression function can be used to produce the stages, MR-CDF must be informed of the proper decompression function to apply at the time of retrieval. When an application stores compressed data in MR-CDF, it is required to *register* an appropriate decompression function with the library, is was illustrated in Figure 4.

A decompression function is an arbitrary procedure which accepts a set of parameters supplied by MR-CDF. These parameters include pointers to the source buffer holding the compressed data and a target buffer into which the decompressed data is to be placed. Additional information, such as the sizes of the buffers and their data types is also provided.

Each time a new multi-resolution variable is defined in MR-CDF, the names of the decom-

pression functions to be used for each compressed data stage must also be supplied. Every decompression function is registered under a particular name. New variables that use the same decompression functions as existing variables may refer to those functions by name.

A decompression function defines a mapping from compressed data to decompressed data. In many cases, it is most convenient to implement the function as a generic piece of code, plus some additional data. For example, vector-quantized data can be decompressed by a simple function which looks-up each code word in a codebook. Changing the codebook changes the decompression function that is being implemented, but the generic code itself need not be modified.

Since this is a common occurrence, MR-CDF allows *auxiliary data* to be stored in addition to the compressed data at each stage. At decompression time, both the compressed data and the auxiliary data are supplied to the decompression function. The advantage of auxiliary data is that common, generic decompression functions need only be registered once with MR-CDF.

# 6 Partial Retrieval

As was illustrated in Figure 2, CDF allows applications to select a portion of the data space, and to retrieve only that part of a variable that lies in the selected region. MR-CDF extends this capability to multi-resolution variables as well.

For single-resolution variables, selections over small regions require less data to be retrieved than selections over larger ones. Ideally, this property should apply to multi-resolution variables as well. However, in some cases it is necessary for MR-CDF to recreate an *entire* variable at the specified resolution even if only a small region was selected. The selection and decompression operations do not always commute.

As we have already described, a multi-resolution variable appears (to an application) to be a collection of $d$-dimensional records. Each stored stage of a multi-resolution variable can also be thought of as a collection of $d$-dimensional records, although the sizes of the dimensions for each stage may differ. In the following discussion, we will let $(n_1, n_2, \ldots, n_d)$ represent the dimension sizes of the variable, and $(n_{i1}, n_{i2}, \ldots, n_{id})$ represent the dimension sizes for the $i$th stored stage. Also, we will define

$$c_{ij} \equiv \frac{n_j}{n_{ij}}$$

as the *compression factor* of the $i$th stage along the $j$th dimension.

Consider the value (in some record) of a multi-dimensional variable at a particular location $(x_1, x_2, \ldots, x_d)$ in the $d$-dimensional space. As we have seen, MR-CDF recreates such values by

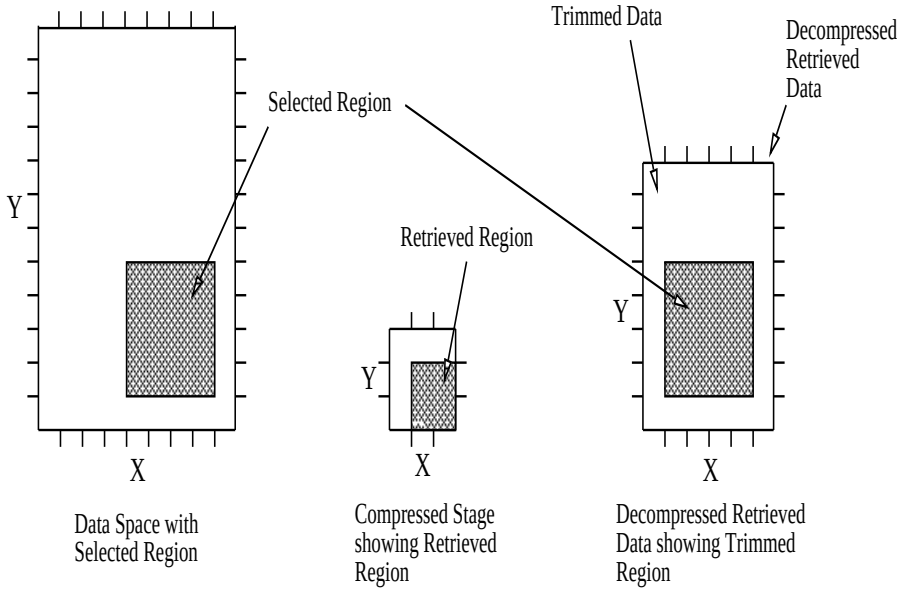



Figure 7: Regular Decompression Example

retrieving, decompressing, and combining the stored stages. We call a decompression function *regular* if the variable's value at $(x_1, x_2, \ldots, x_d)$ can be determined using only the stored values at location $(x_1/c_{i1}, x_2/c_{i2}, \ldots, x_d/c_{id})$ of each stage $i$.

MR-CDF prefers decompression functions that are regular. When the decompression functions for all of a variable's stages are regular, MR-CDF may be able to produce the selected data by retrieving only portions of each compressed stage. In other words, MR-CDF is free to perform selection on the compressed stages *before* decompression.

Figure 7 illustrates regular decompression of one stage of a two-dimensional variable. As the figure illustrates, MR-CDF may need to retrieve only part of the stage to satisfy a query. MR-CDF determines which part of a stage to retrieve by scaling the request according to the compression factors in each dimension. (In the figure, the compression ratios are 3:1 and 4:1 in the X and Y dimensions, respectively.) The figure also shows that the actual volume of data that results from the decompression process may be greater than what the application selected. (This depends on the specific request and the compression factors of the stage data.) MR-CDF incorporates a *trimming function* into its merge procedure to discard any decompressed data that was not requested.

The simple averaging compression technique used in our earlier example (Figure 3) is a regular decompression technique. Many other more realistic and effective techniques are also regular. However, many others are not. Examples of non-regular techniques include the multi-resolution vector quantization described in [MaRe91] and JPEG-like techniques [Wa91] in which

the input data is transformed. In these cases, MR-CDF must decompress entire stages regardless of the size of the selection. The trimming function is then responsible for selecting from the recreated data.

MR-CDF assumes that all decompression functions are regular. However, non-regular decompression is easily accommodated by treating it as a degenerate case of regular compression. Specifically, non-regular decompression can be accommodated by defining each compressed stage to have size one in each dimension. This will force MR-CDF to retrieve the entire stage, regardless of the size of the selection. Defining a stage in this way does not restrict in any way the actual volume of compressed data that can be stored in the stage. Although each stage record will consist of a single "value", the value may be of a composite type, such as an array, of arbitrary size.

# 7 Implementation

To an application, MR-CDF provides a superset of the services provided by CDF. MR-CDF is also *implemented* using CDF. All data storage and retrieval is performed by CDF. MR-CDF acts as a coordinator between a group of CDF archives and the application-specified decompression procedures.

Each MR-CDF archive is implemented as a set of CDF archives. Specifically, a MR-CDF archive is implemented by a single *base CDF* plus a set of *stage CDFs* for storing the compressed data stages. There is a stage CDF for each stage of every multi-resolution variable defined in the archive. This is illustrated (for an archive with a single multi-resolution variable) in Figure 8.

An MR-CDF archive may contain a mix of single-resolution and multi-resolution variables. Single-resolution variables are implemented directly in the base CDF. Requests to store and retrieve such variables are translated into appropriate CDF calls on the base CDF. In addition, the base CDF maintains global information about the MR-CDF archive such as the number of defined variables and their names. It also maintains general information about the multi-resolution variables in the archive.

When MR-CDF receives a request to retrieve a multi-resolution variable, the following steps occur. MR-CDF first retrieves general information about the variable (such as the number of stages that are available and their sizes) from the base CDF. Using this information, MR-CDF then translates its request to a series of retrieval requests on the stage CDF's.

As data is retrieved from each stage, it is placed in a holding buffer and then passed through

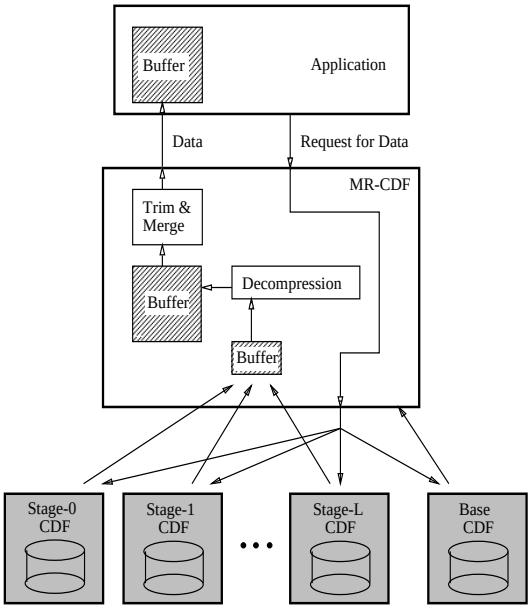



Figure 8: Implementing MR-CDF with CDF

the appropriate decompression function. (Auxiliary decompression information is stored as attributes of the stage CDFs and is retrieved using the attribute/value manipulation facility provided by the CDF library.) The decompressed data is then trimmed and merged with data from the other stages in the merge buffer. To avoid unnecessary copying of data, the decompressed and trimmed stages are merged directly into the application's buffer.

MR-CDF runs on UNIX systems for which CDF is supported. Currently, only the C language interface is available. Since UNIX does not provide a run-time linking facility, it is not possible to define new decompression functions to the MR-CDF library without recompiling it. (New multi-resolution *variables* using existing decompression functions can be added at any time.) However, the procedure for adding new decompression functions is very simple.

# 8 Conclusion

MR-CDF provides an abstract interface to multi-resolution scientific data. Its program interface allows applications to define, store, select, and retrieve data. MR-CDF can make lower resolution data available quickly, allowing applications to trade off resolution for retrieval time.

MR-CDF is implemented using NASA's CDF (Common Data Format) library and runs on any UNIX system supported by CDF. Existing CDF applications can use MR-CDF with minimal modifications.

MR-CDF stores multi-resolution data as a series of compressed data stages which can be

decompressed and combined to produce the data at different resolutions. Retrieval of compressed data introduces a tradeoff between I/O costs and processing costs. Compression reduces the volume of stored data, and therefore the I/O cost for its retrieval. However, the decompression and recombination of the data introduces processing overhead. Technological trends suggest such tradeoffs will become more beneficial with time. The performance of processors continues to improve rapidly, while access times for I/O devices have changed little.

Since CDF utilizes the UNIX file system, distributed operation of the MR-CDF library is possible among machines with access to a common file system, such as NFS. We are currently planning a distributed version of MR-CDF for systems which do not share files.

### Acknowledgements

Thanks to M. Manohar for several helpful discussions, and for providing test data for MR-CDF.

## References

[MaRe91] Markas, A., and J. Reif, "Image Compression Methods With Distortion Controlled Capabilities", *Proc. IEEE Data Compression Conference*, April, 1991.

[RoCa90] Roelofs, L. H., and W. J. Campbell, "Using Expert Systems to Implement a Semantic Data Model of a Large Mass Store System", *Telematics and Informatics*, 7, 3/4, 1990, pp. 361-377.

[ShWa88] Short, N., Jr., and S. L. Wattawa, "The Second Generation Intelligent User Interface for the Crustal Dynamics Data Information System", *Telematics and Informatics*, 5, 3, 1988, pp. 253-268.

[Tr90] Treinish, L., "The Role of Data Management in Discipline-Independent Data Visualization," *SPIE/SPSE Symposium on Electronic Imaging Science and Technology*, February, 1990.

[TrGo90] Treinish, L., and M. Gough, "A Software Package for the Data-Independent Management of Multidimensional Data,", *Eos*, 68, 28, July, 1987, pp. 633-635.

[TiMa90] Tilton, J. C., and M. Manohar, "Hierarchical Data Compression: Integrated Browse, Moderate Loss, and Lossless Levels of Data Compression," *Proc. International Geoscience and Remote Sensing Symposium*, May, 1990, pp. 1655-1658.

[Ti89] Tilton, J. C., "Image Segmentation by Iterative Parallel Region Growing and Splitting," *Proc. International Geoscience and Remote Sensing Symposium*, May, 1989, pp. 2420-2423.

[Wa91] Wallace, G. K., "The JPEG Still Picture Compression Standard," *Communications of the ACM*, 34, 4, April, 1991, pp. 30-44.