NCSA/NARA Technical Report:

Scientific formats for geospatial data preservation
A study of suitability and performance

Mike Folk
mfolk@ncsa.uiuc.edu
Vailin Choi
vchoi@ncsa.uiuc.edu

National Center for Supercomputing Applications
University of Illinois at Urbana-Champaign

January 8, 2004

Abstract

The volume, complexity, variability and accessibility of geospatial of digital geospatial data pose formidable challenges for the National Archives. Scientific and engineering communities faces many of the same challenges in managing large, complex digital data collections. This study examines gridded and vector geospatial data through the prism of scientific data management, with particular emphasis to scientific data formats and software. Geospatial data is characterized in terms of data types, size and storage requirements. Scientific data formats are described, including some of the benefits of using scientific data formats and software. Sample geospatial data types are mapped to corresponding scientific data structures, in particular those supported by HDF5 and HDF-EOS 5. The paper concludes with a report on experiments that are underway to assess storage and access implications of using HDF5 and HDF-EOS 5 for sample geospatial data types that are of particular interest to NARA.

1 Introduction

The preservation of federal geospatial data presents daunting challenges for the many federal agencies, the National Archives in particular. The volume of federally-collected geospatial data is already virtually unmanageable, and promises to grow almost without bound in the future. The variety of geospatial data is very broad, ranging from remote-sensing to mapping to surveying. Its sources range over virtually all federal agencies, its uses cover nearly every aspect of government and every discipline, and new uses are being discovered every day. Furthermore, the technologies that produce geospatial data change constantly. Media, applications, formats, software – all are changing constantly.

The stewards of geospatial data are not alone in facing these challenges. Scientific communities must addresses many of the same issues and concerns, sometimes with the very same data. Problems such as the explosive growth of data archives, the need to access and process enormous volumes of data, the need to access distributed, heterogeneous collections efficiently, and the need to preserve meaning over time command a great deal of attention across virtually all scientific disciplines.

Particular areas of focus include learning to achieve high performance access to data on massively parallel computing systems, developing scientific data packaging formats and software that are robust to changes in computing systems and storage technologies, and finding ways to package and distribute large, complex collections of scientific data to broad, interdisciplinary communities.

The purpose of this study is to investigate how our experiences with scientific data management can help address the challenges of preserving federal geospatial collections

in ways that make the data as accessible and usable as possible for current and future generations.

In section 2 we characterize geospatial data in terms of data types, size and storage requirements. In section 3, we describe scientific data formats, and show how the geospatial data might map to scientific formats, in particular HDF5 and HDF-EOS 5. Section 4 reports on experiments that are underway assess storage and access implications of using HDF5 and HDF-EOS 5 for sample geospatial data types that are of particular interest to NARA.

2 Geospatial data

Federal geospatial data covers a wide variety of themes of importance to federal agencies, their constituencies and collaborators in the private and public sectors, and ultimately to the National Archives.

2.1 Types

Although the number of different kinds of geospatial data is enormous, these types of data fall into a fairly small number of categories. In this study we consider the following categories of geospatial data.

2.1.1 Grid types

We define *grid types* as those whose primary data contents map to an orthogonal grid, usually defined by a Cartesian coordinate system. Gridded data typically describes, at some resolution, a region of space characterized by some property (color, altitude, vegetation). The data points contains value at discrete points in the field, hence the data fields for these types can be represented by simple arrays of two or more dimensions. Geographic map projections are simple examples of this type. The key feature of these types is that the geographic location (e.g. latitude, longitude, depth) of any element can be described as an array coordinate. [1]

With grid types, one or two of the dimensions correspond to geologic location (e.g. latitude and longitude). Other dimensions may represent altitude, depth, or time, or they may correspond to nominal scales, such as spectral bands or geologic layers. Grid types can have any number of dimensions, but for the purposes of this study, we subdivide grid

[1] Much more needs to be said if we are to pin down the fundamental differences between gridded and non-gridded types. For instance, gridded types use a set of discrete points conceptually organized on a discrete grid (and linearized on disk), and yet that is a representation of a continuous field. Each dimension is potentially continuous, but we can't, or don't need to, represent all points. There are efforts (SAF, etc.) to describe these continuous spaces in data structures by storing more extensive topological information, for example interpolation functions at each point in discrete space that are sufficient to compute the behavious of the space within a neighborhood. For most practical purposes, this is not needed, but it could be useful in some cases. As for vector types, they also often represent continuous entities. How important all this is for persistent storage I don't know.

types into four categories: 2D grid, multi-layer 2D grid, and 3D volumetric grid, and swath.[2]

2.1.1.1 2D grid

This category includes data that can be described primarily as an array of simple homogeneous geographic elements, accompanied by attributes that help interpret the meaning and geographic position of the elements.

Examples of 2D grids are the digital elevation model (DEM), the Digital Orthophoto Quadrangle (DOQ), raster images, and the myriad datasets that are represented as 2D earth projections.

2.1.1.2 Multi-layer 2D grid

This includes datasets in which several 2D layers are present and all layers are mapped to a standard 2D earth projection.

Examples include a multi-layer dataset in which each layer is a different theme, but all layers use the same projection. For instance, a dataset consisting of a DOQ and a DEM that cover the same are, have the same spatial resolution, and the same projection. Other examples include multi-spectral and hyper-spectral datasets, and time series' of 2D grids.

2.1.1.3 3D volumetric grids

3D volumes are the 3-dimensional extension of 2D grids – they are arrays in which the geographic position of the elements (voxels) are computable.

Particularly common examples are volumes in which each voxel provides geologic information, such as soil composition. In some cases a 3D volume could also be used to instantiate the same type of data as a multi-layer 2D grid.

2.1.1.4 Swath

In all of the preceding structured types, a formula can be used to find location in an array of an item of interest, and hence only the data fields are stored in arrays. A swath is different in that one or more *geolocation* fields are required to tie the swath to particular points on the Earth's surface. The *HDF-EOS swath* exemplifies this type of structure. As defined in [5] the Swath interface requires the presence of at least a time field (or a latitude/longitude field pair.

The swath is designed to support time-ordered data such as a satellite image consisting a time-ordered series of scanlines.

2.1.2 Non-grid types

We refer to *unstructured types* as those in which the location of the primary data contents are described explicitly, rather than being mapped to a single structure, such as Cartesian

[2] The paper "Storing and Manipulating Gridded Data in Databases" by Barrodale Computing Services Ltd., describes a broad range of uses of gridded data, and also describes a database approach to accessing gridded data. See [2].

array. An unstructured types may also explicitly describe topological relationships among its elements.

In this study, we encounter only one category of unstructured types, which we refer to as vector types. There are, however a number of other important unstructured types that will need to be considered in future studies, such as tabular data, unstructured mesh types, tree-structured types and multi-resolution types.

2.1.2.1 Vector types

A USGS glossary defines vector data as follows:

> *Vector data,* when used in the context of spatial or map information, refers to a format where all map data is stored as points, lines, and areas rather than as an image or continuous tone picture. These vector data have location and attribute information associated with them. [3]

In other words, vector types explicitly describe the location of all points, and, where there are lines and areas, the points that constitute these are also explicitly described.

The range of geospatial data types that are best represented as vector types is very broad. Examples of vector data occur in datasets that store features such as lakes and rivers, political boundaries, and topographic contours.

The attributes referred to in the definition commonly represent many kinds of metadata, such as information about the object that a point, line or area describes, as well as topological information.

[3] http://edcsgs9.cr.usgs.gov/glis/hyper/glossary/index

Figure 1. Some geospatial object types, with examples of each type.

Geospatial object		Characteristics	Geologic examples
Grid	2D grid	Simple homogeneous elements. Single theme.	DOQ, DEM, map
	Multi-layer 2D grid	Collection of multiple 2D grids, possibly with different themes.	Geologic DEMs, one DEM for each of several geologic species. Multi- and hyper-spectral images
	3D volume	Simple homogeneous elements. Single theme.	Geologic volume models with thematic information, such as soil composition
	Swath	One or more layers, each layer a different theme. Location fields required for geo-referencing.	Satellite images such as Landsat, consisting of time-ordered scan-lines
Non-grid	Vector	Primitive types are points, lines, arcs, certain closed shapes, and areas.[4] Topological information often explicit.	Street maps, topographic maps, contoured surface, geologic well log

2.2 Size and storage requirements

2.2.1 Grid types

Since gridded data is commonly collected by an instrument (camera, satellite instrument, etc.), the amount of data to be collected depends on the resolution of the instrument. Because of enormous advances in instrumentation, the resulting volumes can be a very large. The instruments on the EOS Terra satellite, for instance, collect about a terabyte of data per day. Clearly, it is important to find ways to store structured data efficiently, particularly archived data.

The potentially enormous volume of grid types is counterbalanced somewhat by the fact that location and topology are implicit in grid types, saving a great deal of storage space. For instance, it is not necessary to describe explicitly what the neighbors are of a particular feature, as that is implicit in the grid structure.

In addition, a number of techniques are available for improving the storage for grid types. In general they involve some kind of redundancy-reduction, or data compression. Data

[4] What about *volumes*?

compression methods can be lossless or lossy. Lossy techniques can often achieve much better compression ratios, but of course at the expense of lost information.[5]

2.2.2 Vector types

Vector data tends to be less voluminous than gridded data. There are a number of reasons for this. The collection of vector data is often less efficient than that of structured data, as it more frequently requires direct human involvement. Vector objects are frequently one-dimensional (a boundary, a road, a point). And even for the same object, vector data can often represent that object more efficiently than gridded data.

At the same time, because of the need for more explicit geolocation information for each object, the storage space needed for each point (e.g. latitude and longitude) of vector types can be much greater than that required for grid types. Also, since vector data collections often contain many quite different types of objects, the associated attributes can also vary greatly. Thus, in comparison with grid metadata, which typically applies to an entire grid and not to the objects within the grid, the storage space needed for vector metadata can be comparatively large.

As with gridded data, compression techniques can be used to reduce redundancy with vector data. However, the nature of the two types of data are quite different, so the techniques that work well for one will not necessarily work well with the other.

2.3 Access requirements

One of the most daunting challenges in thinking about how to archive geospatial data is to anticipate how the data will be accessed. Patterns of access vary enormously, and it is impossible to know how future generations will want to access the data that we archive today. The best we can do is to make some educated guesses. The importance of guessing correctly, or at least reasonably correctly, comes from the fact that what we store today, and how we store it, can have a huge influence on the usability and accessibility of that data at a future time.

Sequential and random access. Access can be divided into two broad categories: sequential and random. *Sequential access* occurs when we read a collection of records in the order in which they are stored, usually beginning with the first record. When sequential access is required, we can access a given record only by first reading through all of the records that precede it in the sequence. *Random access* (or "direct access") occurs when we read one or more records directly without having to access the records that precede it.

Sequential access is fast and simple as long as you need to access all information in the same order, but it can be very inefficient if only a small subset of the data is needed. For instance, suppose you have a vector dataset with 20,000 features. If you wish to read all

[5] We are not aware of the extent to which NARA and others have investigated issues of "lossy" storage technologies. In many scientific applications, some loss of information is quite acceptable. See for instance, the paper "Multi-Resolution Modeling of Large Scale Scientific Simulation Data" [1], in which it is argued that a wavelet compression technique can capture the important physical characteristics of the target scientific data. This may be another interesting area of research for NARA.

of the features into memory to display them, sequential access is the best. However, if you need to access only a small number of features, sequential access can be intolerably slow.

Gridded data is also accessed both sequentially and randomly. A very common type of random access of gridded data is subsetting, where some subset of the elements in an array are read or written. In the geospatial world, subsetting can take many forms. For instance, we may be interested in a small rectangular geographic region of a large image, the same region from several different layers in a multi-layered grid, or a 3D region from a geologic volume. The region of interest may be specified by an irregular shape, such as a city boundary. Another type of subsetting is to retrieve those parts of a dataset that satisfy certain criteria.

Parallel access. An increasing number of high end systems can I/O in parallel, and have parallel file systems, and it is important that data is organized in a way that facilitates parallel I/O. In many instance, it is possible to divide a large geospatial dataset into regions that can be processed in parallel, and in such cases it is important that the corresponding file structures and I/O software facilitate this by allowing random parallel access to data. Indeed, as the sizes of datasets increase over time, it may well become essential that parallel I/O be possible.

Distributed access. In the past, when a particular dataset or collection was needed, it has been common to move the entire dataset or collection to the place where it was to be used. We are already at a point where this is not reasonable, and in some cases may not even be possible. Rather, data grids are emerging as a way to provide access to data that is stored somewhere else. One important capability of data grids is the ability to provide partial random access, access not just to whole files, but to objects within files, and portions of objects. File structures and I/O software that facilitate this kind of access will be increasingly important.

3 Scientific formats, geospatial data preservation and HDF5

In this section we describe common characteristics of scientific data formats that have potential benefit for the geospatial community. This is followed by a description of HDF5 and HDF-EOS 5, the formats that have been investigated in this work.

3.1 Characteristics and features of interest for geospatial data preservation

General purpose scientific formats typically have many features of potential value for archival storage. In this section we identify some of those features and how the contribute to effective long-term digital preservation.

3.1.1 Open documentation and code

When we use the term “open” scientific formats, we refer to formats designed for scientific applications whose structure is in the public domain. Most scientific formats are also open in the sense that the formats can be used with little or no licensing restrictions, and free software is available for reading and writing data in the format.

Open documentation of digital formats is particularly important for long-term digital preservation. Without open documentation there can be no way to understand, therefore access, the information.

3.1.2 Widely used for geospatial data

Many open scientific formats, such as netCDF [9], HDF4 [6], HDF5 [7], and FITS [4], are already widely used with certain types of geospatial data, particularly that data that arises from the communities that employ these formats. Because much of this data, and the software that accompanies it, is useful beyond just the originating communities, there is a natural desire to make these formats work well with geospatial applications.

In particular, as we have worked with HDF4 and HDF5 applications in several geospatial communities, most notably the Earth Observing System, we have noted very strong conceptual ties between HDF applications and those of the broader geospatial community. In addition, through HDF5, HDF has evolved a comprehensive set of features that address many of the most critical challenges of long-term preservation for science and engineering data.

3.1.3 Basic data types and structures.

Most general purpose scientific formats include data structures such as arrays for supporting large structures, a variety of datatypes (integers, floating point numbers, strings) for supporting varied of information types, and ways to accommodate metadata. These building blocks of digital information are critical to long-term preservation of scientific digital information, including geospatial data. We examine file structures and data structures in more detail below.

3.1.4 Self-description.

Some scientific formats are also "self-describing" in the sense that they incorporate, within the file itself, enough information that available software can know whatever it needs to in order to effectively access and use objects in the format. Of course, in the context of long-term preservation, this feature requires the evolution of I/O software. Nevertheless it can be an effective way to preserve the full meaning of data long after the producers of the data have disappeared.

3.1.5 Portability.

Many scientific formats are portable in two senses. (1) They usually can store data in such a way that it can be move from one computer architecture to another, optionally without any loss of information content. This is very important for long-term archiving, because it increases the likelihood that future technologies will find collections accessible. (2) Their access software is designed to move easily among many different computing platforms – different programming languages, operating systems, and computing architectures. This is also important for long-term archiving because it increases the likelihood that access software can evolve into the future.

3.1.6 Scalability.

Some formats also provided features to enhance scalability, both in terms of data volume and in terms of data movement. The ability to store very large granules and large numbers of individual objects efficiently and safely is increasingly important to preservation activities, particularly those that organizations such as NARA face.

3.1.7 Random and partial access.

Equally important is the ability efficiently to write, read, and find objects, as well as the ability to perform random access to individual objects or records in a collection, and partial access to aggregate objects. In some cases, the I/O software for scientific formats is tuned for operating in high performance environments, such as massively parallel computing systems. Such systems are rare today in preservation environments, but as the digital revolution proceeds, they may well provide the only means with which to deal with the enormous volumes of data that digital archives must manage.

3.1.8 Collection heterogeneity.

Some formats also provide the ability to organize meaningful collections of heterogeneous objects, which can be very important for archival data management of complex combinations of data objects such as we find increasingly in applications involving geospatial data.

3.2 File structures and data structures

Computer scientists have worked hard to develop data structures and techniques that facilitate efficient access of structured and unstructured types. These techniques can differ significantly, depending on whether the data is structured or unstructured, and also depending upon the types of access that at expected. It is important to understand what the most likely patterns are for a given collection, so that the most suitable storage structures and access software may be used.

Storage efficiency versus access efficiency. Access requirements can have a major effect on storage requirements. It is generally the case that faster or more flexible access requires extra storage. For example, extra index structures can be built, at a cost of extra storage, to speed up access to individual records in a collection. Also, there are special data structures that simultaneously permit efficient sequential and random access to data, and not surprisingly such data structures require extra storage space.

Lazy versus eager processing. It is also generally the case that extra pre-processing is required to create structures to facilitate finding and accessing records. Building an index takes extra computing time as well as extra storage space. If random access is frequent and needs to be fast, this extra time and storage can be well worth the time and space to build needed data structures at the time the data is brought into an archive. This is sometimes referred to as *eager processing.* But if such access is rare, it may be better to save the time and space until the search is needed (*lazy processing*).

Since in many cases it will be impossible to predict how today's data will be accessed decades and centuries from now, we must always to insure that it can be converted to a form that will support the kinds of access that will be required.

Some useful structures. Although the types of data structures that are available to support efficient access are too varied to cover in this paper,[6] general purpose scientific data formats provide some useful structures for improving storage and access to geospatial data. Two good examples are data compression and chunking.

Compression is a well-understood and heavily used method of decreasing storage by reducing redundancy in data. Some general formats, such TIFF and HDF, support compression of individual objects, or even parts of objects. Compression not only improves storage, but can also improve I/O because there are fewer bytes to move between one location and another. For example, a recent experiment with the HDF5 format demonstrated that compression can improve access time particularly well when data is stored distributed on a network.

Chunking occurs when a large multi-dimensional array is organized as a collection of independently stored sub-arrays. The 2D version of chunking is called tiling, and results when an array or image is stored as a set of individual tiles. Chunking can local the storage of geographically local pixels in an image, or voxels in a volume, and as a result can improve performance of partial access. Chunking can also be used to create array structures that can grow dynamically in any direction, a capability that may prove useful for time-series data.

When chunking and compression are combined, a partial I/O operation requires only that the chunks containing the region of interest be loaded and uncompressed. For very large images, the resulting improvements in I/O efficiency can be enormous.

Geospatial vector data is different. Scientific formats generally are designed for data with a great deal of structural repetition, such as a large raster image, a geologic volume or a triangle mesh. Datasets containing many records (images, etc.) generally contain *fixed length* records. In contrast, geospatial vector data typically contains *variable-length records* – that is, objects in a single collection that can vary enormously in size. A large shape file can contain shapes ranging for one point in size to many thousands of points. Because of this, there is no simple one-size-fits-all data structure that can provide good storage efficiency and good I/O performance.

3.3 Format generality versus specialization

General purpose scientific formats are typically designed for a range of uses, and as a result may not provide optimum performance for many specialized applications. As a result, a format designed with a particular application and data in mind may outperform a more general format in important ways. This advantage of a specialized format must be weighed against the cost of developing and maintaining the two types of format.

A general purpose format such as TIFF, netCDF and HDF5 is likely to have a broad constituency, so that the cost of developing and maintaining it can be amortized across a broader funding base than perhaps a specialized format. Furthermore it may incorporate

[6] It would be a good idea to survey the range of available data structures in a future study. In particular geospatial structures designed to provide good partial access, such as R-trees and oct-trees. It may be possible to formalize and standardize some of these to the extent that they could be used for long-term preservation.

features the a special-purpose format does not have, that may prove to be valuable, and perhaps even necessary, as applications evolve. We have seen this, for example, in the evolution of computing systems from single processor (with sequential file systems) to cluster computing with parallel file systems. The HDF5 format and software work well on many parallel systems because a small part of the initial HDF5 user community needed this kind of access. Similarly, the cost of supporting features such as chunking data compression may be more feasible with a general-purpose format.

Another way to view the differences between general purpose and specialize formats is that with general purpose formats, object specialization is accomplished by restricting the semantics and methods of more general objects. For example an array (very general) is interpreted as an image (specialized) with certain semantics and method. In object terms, a more specialized object can inherit the semantics and methods of the more general objects. This inheritance can have enormous advantages in terms of software engineering, because it may permit the higher levels to re-use methods at the lower levels.

Of course, just as a stock car will never win the Indy 500, a general purpose format will never be as perfect for an application as a specialized format (and I/O software) for which all necessary resources are available to make it do everything that is required. And if the specialized application is important enough, the special format should be supported.

3.4 What is HDF5?

HDF5 [7] is a general purpose format designed particularly for scientific and engineering data, together with a portable I/O library designed to facilitate high performance data access. The HDF5 format has only a few primitive objects, but these objects can be used to represent images, multidimensional arrays and tables, as well as more complex structures.

The HDF5 format and software are open and free for any use, both commercial and non-commercial. There is currently just one HDF5 I/O library implementation, developed and licensed (free for all uses) by the University of Illinois.

Since HDF5 is a very general format, it is possible to use it in many ways for a given type of data. For this reason there is an emphasis within the community on standardizing the way HDF5 files are organized. This is done typically in two ways: (1) by publishing standards on the meaning and organization of objects vis a vis a particular application area, and (2) by providing a reference API and library implementation that facilitates and enforces the standards.

HDF5 is of particular interest for this study because it contains all of the capabilities described above in some useful form. HDF itself grew out of a need to store and share heterogeneous scientific data among many computing platforms, and to access, visualize and analyze scientific data easily and efficiently. HDF5 was designed to provide all of the capabilities of HDF, but in a way that could scalable to meet future demands of collection size and high end computing.

HDF5 file. An HDF5 file is a container for storing scientific data. At the simplest level, an HDF5 file can contain two primary objects, groups and datasets.

A *group* is similar to a UNIX directory or folder – it is a way to group other objects within an HDF5 file. The grouping structure in an HDF5 file is very general (e.g. groups can share objects), and hence makes it possible to organize the objects in an HDF5 file in complex ways.

A *dataset* is a data array with metadata. By "data array", we mean an ordered collection of identically typed data elements distinguished by their indices. HDF5 supports a very general range of data element types (datatypes), including the usual integer and floating point numeric type, string types, compound (record-structured) types, and user-defined types.

Datasets can also contain *attributes*, which are user-defined metadata. The type of an attribute can be as complex as a dataset array.

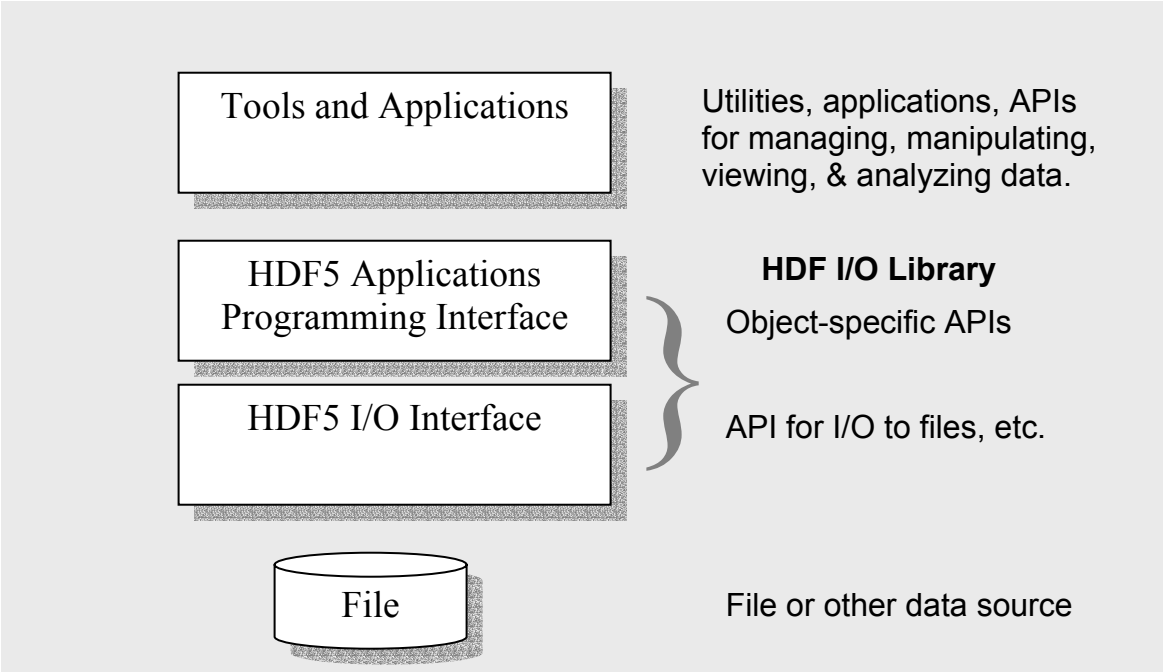
Datasets also have associated special storage options, which give an HDF5 application flexibility in determining how to store the HDF5 data array. Storage options include compression, tiling (chunking), and extensibility. These options are designed to promote storage and I/O efficiency.

Another useful feature of HDF5 is a *user block*, which is a block stored at the beginning of an HDF5 file that can hold any data that a user wishes. The user block was included in the design of HDF5 precisely for applications such as archiving, so that extra information could be provided to help future users decipher and interpret the contents of the accompanying file.

HDF5 software. Figure 2 illustrates the software environment in which HDF5 is typically used. As with many general purpose scientific data formats, there is an I/O library that reads and writes data in the format. A typical application invokes this library. The application may be a visualization tool with a graphical user interface, it may be a utility that performs some analysis on the file, or it could be an application that accesses one or more of the objects in the file. Some applications, such as HDF-EOS, are just other libraries that themselves provide APIs but in the process impose a specific view of the objects in the file.

The key to HDF5 software is the HDF5 I/O library, which is designed to support scalable scientific data management in a variety of computing environments, ranging from personal workstations to high end computers. Naturally, the library mirrors the format, and as such provides the ability to create complex information structures, to access data efficiently and flexibly (including complex subsetting), to store data efficiently, and to perform I/O effectively and flexibly. The HDF5 library supports key language models by providing application programming interfaces (API) for several languages, including C, Fortran, Java and C++.

Figure 2. HDF5 software environment.



3.5 HDF-EOS for digital preservation

HDF is also of interest because of it's use in the geospatial arena. An HDF-based NASA standard called HDF-EOS is used to manage very large (and growing) collections of geospatially referenced data generated by NASA's Earth Observing System. HDF-EOS specifies standard ways to organize HDF files that map well to the many data types that result from NASA's Earth Observing System (EOS) missions. This passage from the "HDF-EOS Primer"[8] describes HDF-EOS.

> To bridge the gap between the needs of EOS data products and the capabilities of HDF, the ECS Project has developed extensions of HDF, which standardize the conventions for writing HDF files, and are called HDF-EOS. These extensions facilitate the creation of Grid, Point and Swath data structures. These structures are composed of native HDF objects and are therefore objects themselves. In the text below, Grid, Point and Swath structures are described in more detail.
>
> The software interface for the HDF-EOS implementation is very similar to the HDF interface. The HDF-EOS interface is used to access the Grid, Point and Swath data structures created by the HDF-EOS library. The plain HDF interface is not used to access Grid, Point and Swath structures. See HDF-EOS Users Guide for the ECS Project and references.
>
> The *Point* interface is designed to support data that has associated geolocation information, but is not organized in any well-defined spatial or temporal way. The *Swath* interface is tailored to support time-ordered data such as satellite swaths (which consist of a time-ordered series of scanlines), or profilers (which consist of a time-ordered series of profiles). The *Grid* interface is designed to support data

> that has been organized in a rectilinear array, based on a well defined and explicitly supported projection. [7]

Many current EOS users find themselves dealing not only with geospatial data in HDF-EOS, but also with many other kinds of geospatial data. Because HDF-EOS combines the features of HDF with geolocation information, other earth science metadata, and standardized geographical subsetting, it seems to be a particularly well-suited scientific format for our study of scientific formats for long-term preservation of digital geospatial data. HDF-EOS 5 is attractive for several additional reasons:

a. HDF-EOS is a widely used, NASA-supported data model and library.

b. The HDF-EOS grid datatype in particular is attractive, as it supports a number of USGS standard map projections, and hence has a high likelihood of mapping well to projection-gridded geospatial data.

c. The HDF-EOS format has the ability to define additional dimensions, thus supporting the vertical dimension needed for some geologic datasets.

d. HDF-EOS 5 is scalable, having been implemented using the HDF5 format.

e. HDF-EOS provides access methods and tools to facilitate meaningful access to geospatial grids. For instance, the HDF-EOS library can convert between a variety of map projects, supporting (at least to some extent) the need to combine grids that use different projections.

f. There is a large and growing base of software for storing, querying, accessing, viewing, and analyzing HDF-EOS data. At the heart of this software is the HDF-EOS API, an implementation of which is supported by the EOS project

g. HDF-EOS also is a good candidate for persistent storage. Its enormous constituency (more than 1.6 million users) and NASA backing make it likely that migration technologies will be available for data stored in the HDF-EOS format.

There are two versions of HDF-EOS: HDF-EOS 2 is based on the older version of HDF (HDF4), and HDF-EOS 5 is based on HDF5. Their data types and access methods are very similar. For this study, we focus on HDF-EOS 5.

4 Experiments with HDF5/HDF-EOS 5

In Section 2 we list the types of geospatial data that are the focus of this study. In mapping these types to HDF5, we had three major considerations:

1. What object types, if any, are available in HDF5 or its derivatives (e.g. HDF-EOS) that can best capture the full information content of each of these geospatial types?

[7] Taaheri, "An HDF-EOS and Data Formatting Primer for the ECS Project", p. 3-1.

In those cases where proper object types are identified:

2. How might these types be organized within HDF5 to facilitate anticipated access requirements?
3. What storage efficiencies are to be gained or lost by converting this data to HDF5.
4. What performance efficiencies are to be gained or lost by converting this data to HDF5?

4.1 Mapping to HDF5 and HDF-EOS 5 object types

In the study, a number of geospatial types was examined to determine the HDF5 object types that can best capture their information content. Figure 3 gives a summary of results. The following sections describing these results in more detail.

Figure 3. Mapping geospatial objects to h5 and its derivatives.

Geospatial object		Formats investigated	HDF-based formats
Structured	2D grid	USGS DOQ. GeoTIFF[8]	HDF5 image/2D array, HDF-EOS grid
	Multi-layer 2D grid	RockWorks[9]	HDF-EOS grid
	3D volume	Rockworks	HDF-EOS grid
Un-structured	Vector	ESRI shapefiles	HDF5 combinations of dataset with compound, variable-length datatypes and/or arrays.

4.1.1 2D grid

2D grids consist of simple homogeneous arrays, plus metadata. For the test case, the Digital Ortho Photo Quadrangle (DOQ) was chosen. DOQs are found in a number of formats, including a "USGS" format, a form of the TIFF format called GeoTIFF, and SDTS. We mapped both USGS and GeoTIFF DOQs to HDF5.[10]

DOQ images map easily the HDF5 "image" format, and this was used in both cases. As for the metadata, the USGS and GeoTIFF DOQ formats differ – USGS uses a standard header, while GeoTIFF has both structural metadata ("tiff header") and a geospatial

[8] GeoTIFF is a TIFF-based interchange format for georeferenced raster imagery. For more information, see http://remotesensing.org/geotiff/geotiff.html.

[9] RockWorks is a product of RockWare, Inc. An ASCII-based format used by RockWorks was used in this study.

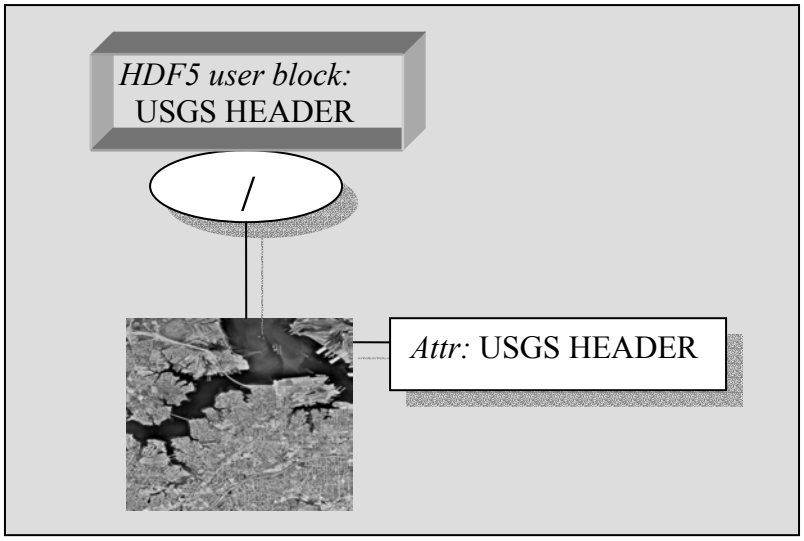
[10] We might also have mapped this data type to the HDF-EOS 5 "grid" type. In fact, we did this to the multi-layer 2D grid, described next.

metadata ("GeoTIFF header"). There are many ways to map these metadata to HDF5 structures.

For this experiment, we chose to map the USGS DOQ metadata to two HDF5 structures : the user block and to an image attribute with the named "HEADER". That is we stored identical copies of the metadata in two places, in both cases using the same ASCII text format that it occurs in the original USGS file. The use of the user block makes all of the metadata available to any application that can read text, a particular benefit for long-term archiving. The use of the image attribute makes it clear that this metadata applies in particular to the image.

For the GeoTIFF DOQ format, we could have used the same approach, but because there are two types of metadata, we chose to use image attributes for both metadata components, respectively named "geotiffheader" and "tiffheader".

Figure 4. USGS DOQ stored as image in HDF5 file. Headers stored as attribute.



4.1.2 Multi-layer 2D grid

Multi-layer 2D grids are similar to 2D grids, except there are many of them in a single container. For this part of the study we chose a geologic grid file in the RockWorks format.[11] The dataset contains 13 elevation models corresponding to 13 geologic layers, each gridded to the same projection.

In this example we chose a multi-layer HDF-EOS 5 "grid" as the target HDF5 format to map to. An HDF-EOS grid contains a series of data fields of two or more dimensions that is mapped to a particular projection. Hence it was straightforward to map each of the 13 elevation models to an HDF-EOS grid data field.

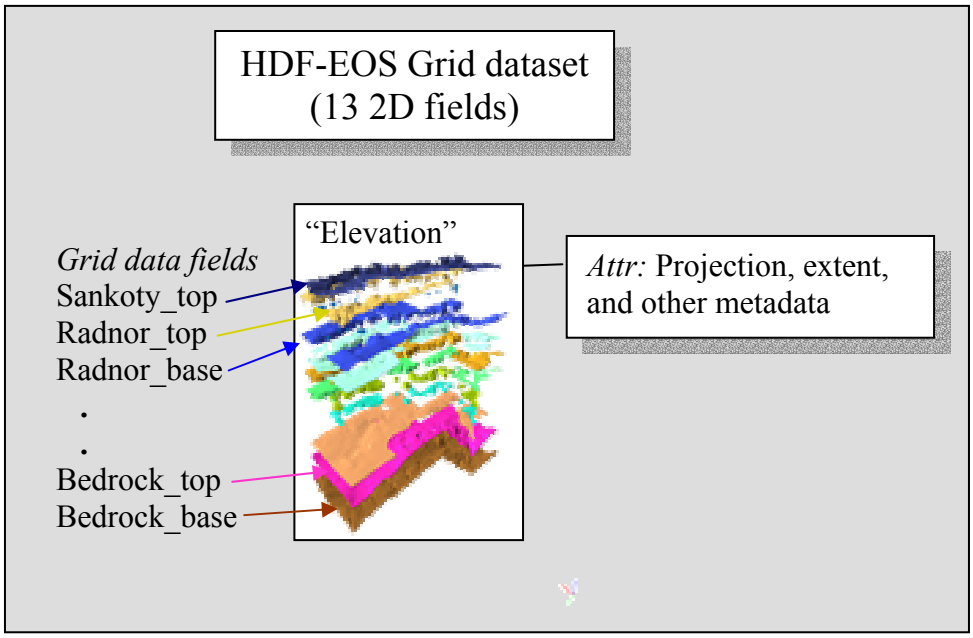
[11] RockWorks is a collection of earth science software that provides access to analytical and visual tools. RockWorks has its own format for gridded data.

The metadata in the RockWorks file mapped easily and naturally to the HDF-EOS 5 grid metadata, which in this case is relatively simple.

The use of the HDF-EOS grid for multi-layer 2D grid is illustrated in Figure 5.

It should be noted that the example chosen for this experiment used the same projection and extent for all layers, and hence the original dataset mapped to a single HDF-EOS grid. If the different layers had had different extents, or used different projections, it would still have been possible to store them in one file, because HDF-EOS permits the storage of multiple grids per file.

Figure 5. Multi-layer 2D grid organized as HDF-EOS grid with 13 data fields.

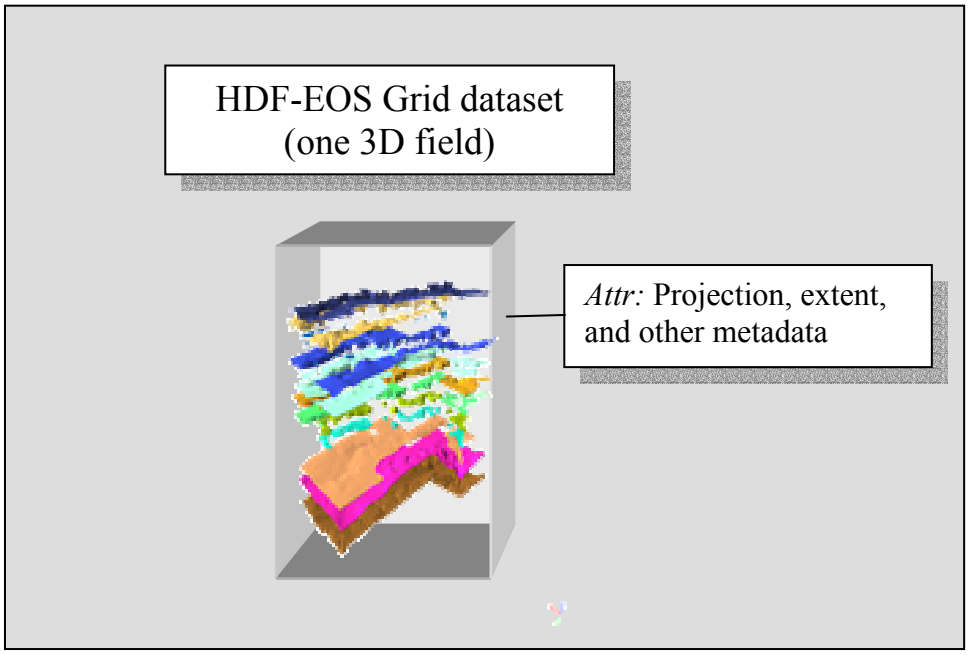


4.1.3 3D volume

As defined above, a 3D volume is just an extension of a 2D grid to a third dimension, such as altitude, depth, or time. For this study, we also used a dataset in a RockWorks format, this format designed for 3D volumes. This dataset provides the same basic information as the one used for the multi-layer 2D grid, but in this case each of the 13 layers is represented by a particular value. Each voxel in the 3D array stores either a fill value (if none of the 13 layers is present at that point) or one of the 13 values.

Since the HDF-EOS grid can accommodate volumes, it was equally straightforward to map the original RockWorks geologic dataset to an HDF-EOS grid, including the metadata.

Figure 6. 3D volume organized as HDF-EOS grid with one 3D data field



4.1.4 Vector

Vector data represents the most challenging data type encountered in this study. In this study, we focused on vector datatypes that do not include topological information. For this we used sample data from a number of ESRI "shapefiles". An ESRI shapefile is described in [3] as follows.

> A shapefile stores nontopological geometry and attribute information for the spatial features in a data set. The geometry for a feature is stored as a shape comprising a set of vector coordinates.
>
> Because shapefiles do not have the processing overhead of a topological data structure, they have advantages over other data sources such as faster drawing speed and edit ability. Shapefiles handle single features that overlap or that are noncontiguous. They also typically require less disk space and are easier to read and write.
>
> Shapefiles can support point, line, and area features. Area features are represented as closed loop, double-digitized polygons. Attributes are held in a dBASE ® format file. Each attribute record has a one-to-one relationship with the associated shape record.

A shapefile is actually typically three separate files with the same base name, but distinguished by the extensions .shp (geospatial data file), .shx (index file), and .dbf (attribute data file). The basic shape information is stored in the .shp file. The .shx file is an index to the shapes in the .shp file, enabling an application to achieve extremely fast random access to individual shapes. Similarly, the .dbf file enables very fast random access to shape attributes.

The study considered a variety of different shapefiles, varying in sized from about a kilobyte to 20 megabytes, in the number of features (from 1 to more than 11,000), and in the complexity of features (from shapes consisting of a few vertices to a shape with more than 38,000 vertices). Figure 7 lists the files considered, with this information.

Figure 7. Information about shapefiles used in study.

Shapefile name	Size (bytes) (.shp+.shx+.dbf)	Total # of shapes	Total # of vertices	Max. # of vertices for any shape in file
A	1,395	1	66	66
B	11,553	44	191	12
C	191,606	219	9,397	1,632
D	3,161,040	2,253	179,106	38,725
E	13,124,370	11,576	721,123	500
F	19,774,790	8,877	1,140,460	500

We began with the assumption that vector data could be stored using the structures and metadata similar to those used for so-called unstructured grids, such as those used in computational fluid dynamics (CFD) and finite element modeling, which are supported in HDF5 by the "HDF5mesh" API and format. This assumption proved misguided. Although it is *possible* to store vector data using these structures, there are important differences between these classes of data.

- The metadata is much different. Vector data contains metadata about topology and geographic entities that is irrelevant in the CFD and similar applications.
- Types of objects differ: In geospatial vector data, the variety of shapes is large, whereas unstructured grids typically consist of large numbers of a few basic shapes.
- Access and query requirements are quite different between geospatial and computational mesh data.

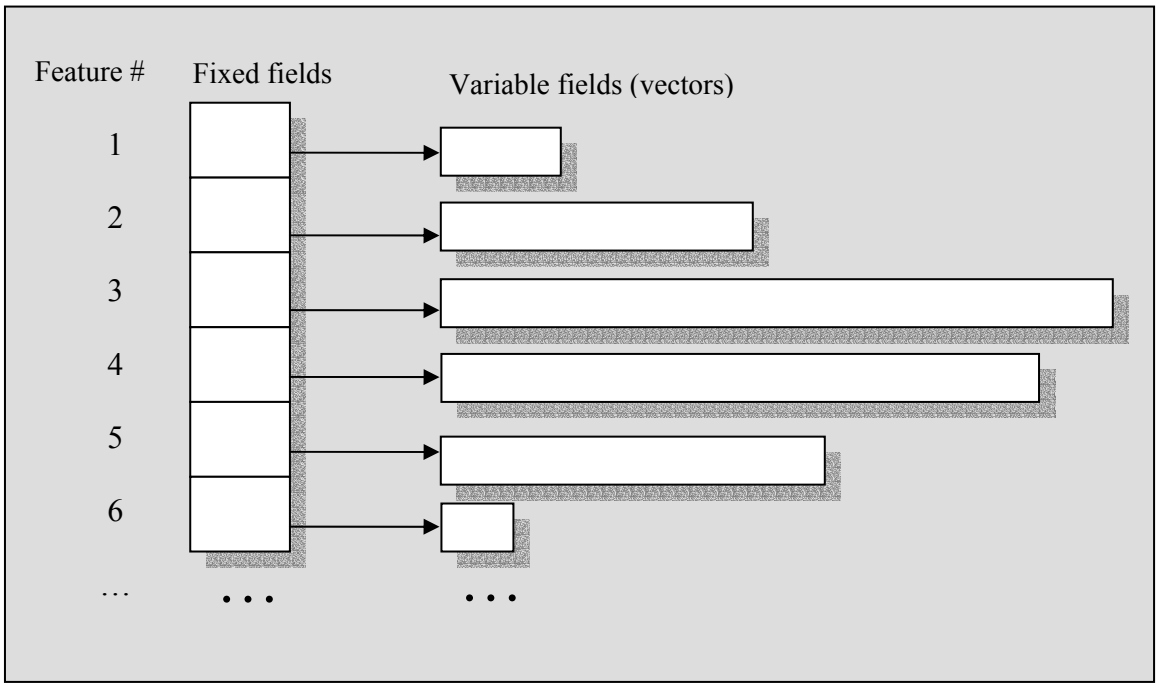
Because of these differences, we chose not to convert vector data into HDF5mesh.

A second approach involved mapping vectors in the most conceptually natural way to HDF5 – using compound data types. This involved creating, for each vector-based feature, a compound HDF5 datatype containing all of the information about that object

that would be contained in a record of the corresponding shapefile. Each resulting object was stored as a rather complex datatype with a number of variable-length fields (Figure 8). Because of the self-describing nature of HDF5, the resulting information was very straightforward to store and access. At the same time, the overhead required to store these records, particularly the variable-length fields could be extremely high. HDF5 uses 32-bytes of overhead to store a variable-length record, so a 2D line segment that might be described with 16 bytes could require as many as 32 bytes of additional overhead. The overhead problem is compounded by the fact that any given feature might include a number of variable-length data types.

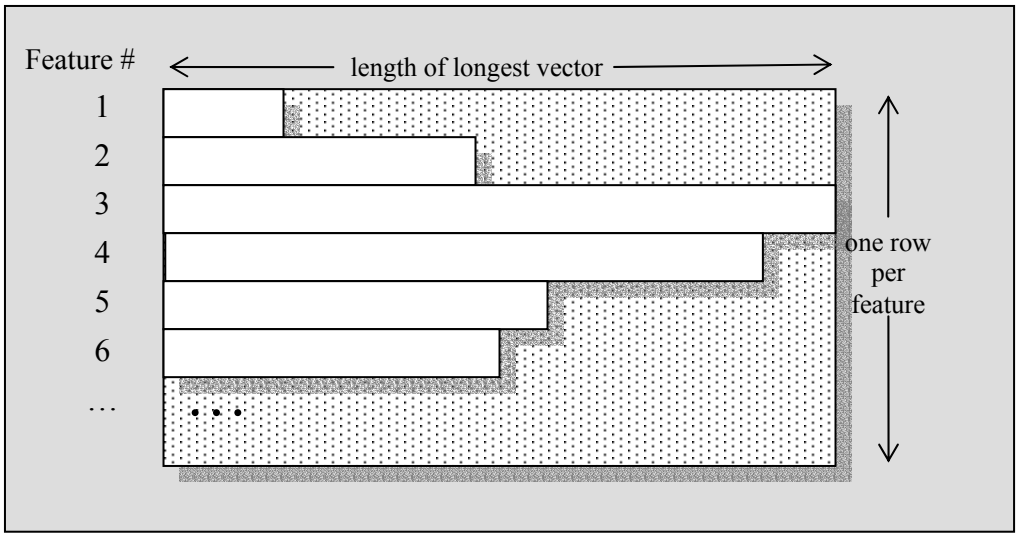
Variations on this approach were also investigated, with some improvement in storage and access time, but in the best case the storage requirements for HDF5 were quite high compared with those of shapefiles.

Figure 8. Second approach. Features stored as a 1D array of variable-length records, one record per feature. Requires substantial overhead for describing each record. This organization is later referred to as format "HA".



A third approach was to store the fixed metadata in a separate structure (hence requiring little overhead) and the variable length data in a large array, with one row per feature. That is, each row of the array contains all of the points that describe a given feature. The array is created so that the row length is large enough to contain the feature with the most points. Zeroes are use used to occupy unused space in an array. This approach is illustrated in Figure 9. The overhead for this structure is even larger than in the earlier case, but in this case data compression might be used to decrease the overhead.

Figure 9. Vector points stored in 2D array, row size equal to longest vector. The shaded area is filled with zero values. Overhead for each row is small. Later in the paper this is referred to as format "HB".



4.2 Conversion

Once each geospatial datatypes was mapped to HDF5 (or HDF-EOS 5), utilities were created to convert the data from its native format to the target HDF-based format. For the most part this was not difficult. In some cases the original format was quite simple. In others (GeoTIFF and shapefiles), software was available for accessing the data in the original format, and the primary task was just to write it out in HDF5.

4.3 Storage results

For each of the geospatial types, we examined the storage implications of converting them to HDF5 or HDF-EOS 5. The differences in storage requirements varied considerably between grid and vector data.

4.3.1 Grid-based formats

Figure 10 and Figure 11 show the storage requirements for sample grid files converted from their original formats to HDF5-based formats.

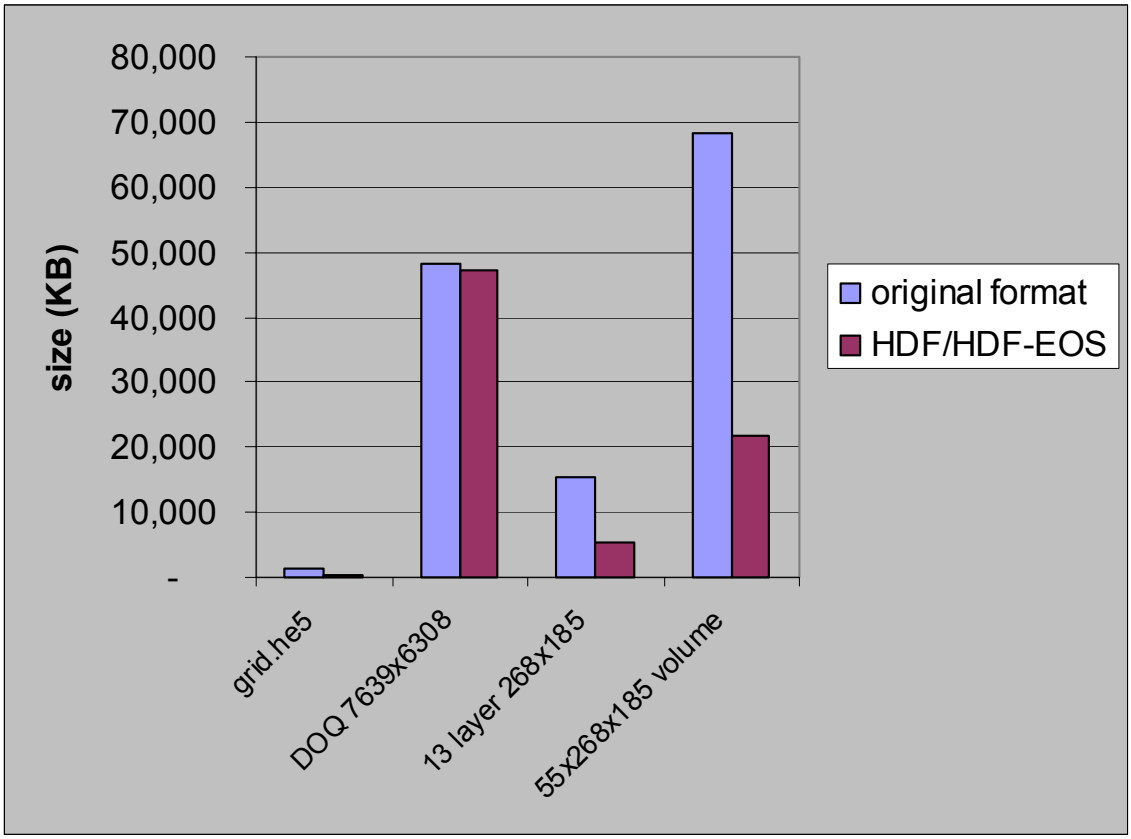
In the case of grids, the storage needs were either similar or much less for the HDF5-based formats than for the original formats. The significant gains occur for the RockWorks formatted files. There is nothing magic about these results – RockWorks uses a text format for both formats described here, and binary formats are almost always better for text.

These studies did not explore the use of data compression in HDF5. Certain files, such as the 3D volume solid model file, would likely lend themselves very well to data compression.

Figure 10. File size comparison -- original format vs. HDF5-based format.

Geospatial object	Description of object(s)	Native format storage requirement		Size (KB) -- HDF-based formats
		Format	Size (KB)	
2D grid	DOQ 7639x6308	USGS	48,203	48,193
	GeoTIFF color DOQ	GeoTIFF	75,000	75,004
	2D geologic grid	RockWorks grid (text)	1,239	440
Multi-layer 2D grid	13 layers of 268x185 grid (64-bit float)	RockWorks grid (text)	15,470	5,202
3D volume	3D volume 55x268x185 (64-bit float)	Rockworks/ solid model (text)	68,172	21,861

Figure 11. File size comparison -- original format vs. HDF5-based format.



4.3.2 Vector-based formats

The storage results for the vector-based format were quite different from those of grid-based formats. As described above, the variability of individual features and metadata do not match up nearly as well with scientific formats such as HDF5 as do the grid formats. As described above, a number of different HDF5 organizations were investigated. The following two provide good examples of the results.

HA Features stored as a 1D array of records, one record per feature, with variable-length fields per record corresponding to the variable sized features. Requires substantial overhead for describing each record. (See Figure 8.)

HB Fixed information stored in one compound type, but one or more 2D arrays used to store the x,y,z, and m. The dimensions of this array are "max number of shapes" x "max number of vertices". (See Figure 9.)

As described in Figure 7, six different shapefiles were investigated. The result of directly converting each to HDF5 are shown in Figure 12 and Figure 13. It will be noted that in all cases, the resulting HDF5 file is substantially larger than the corresponding shapefile. In the case of shapefile D, file HB is nearly 200 times larger.

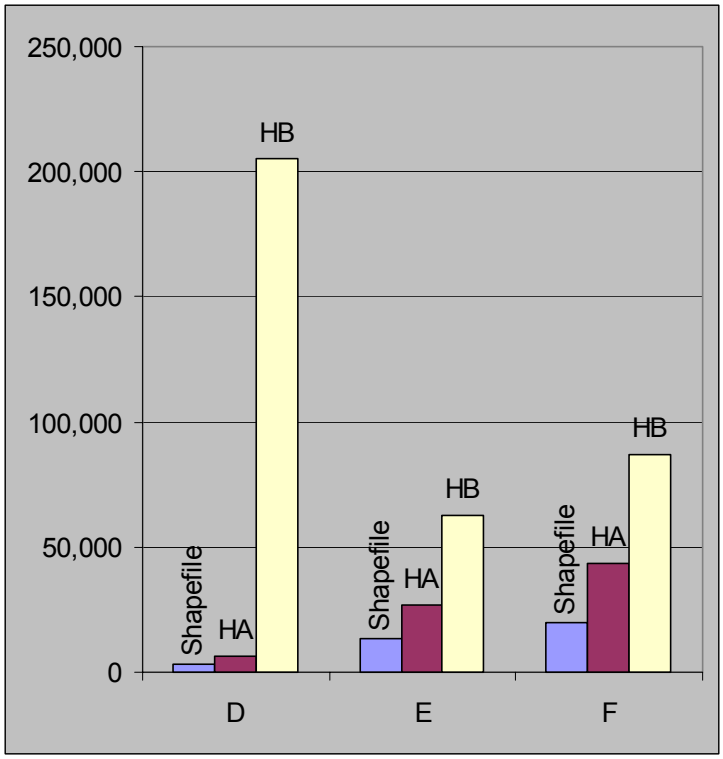
The differences in file sizes are due to two things: in the case of HA and HB, the overhead of storing variable-length data in a general-purpose format such as HDF5 can be extremely high. This suggests that we look for ways to efficiently store this kind of

data in future format development. It also raises the question of what kind of storage improvements can be obtained through the use of the data compression features of HDF5, which we examine next.

Figure 12. File sizes resulting from direct conversion of shapefiles to HDF5.

Shapefile name	Total # of shapes	Max. # vertices for any shape	Size (KB)		
			Shapefile	HA	HB
A	1	66	1	10	67
B	44	12	12	26	25
C	219	1,632	192	376	1,345
D	2,253	38,725	3,161	6,411	205,186
E	11,576	500	13,124	27,114	62,535
F	8,877	500	19,775	43,380	86,930

Figure 13. File sizes resulting from direct conversion of shapefiles to HDF5. (Shows large files only.) Note HB for file type D, a 100-fold increase in file size due to the presence of a single feature with a very large number of vertices (38,725).



Data compression. We next investigate the possible effect that data compression can have on storage of vector data in HDF5. There is, after all, a great deal of redundancy in

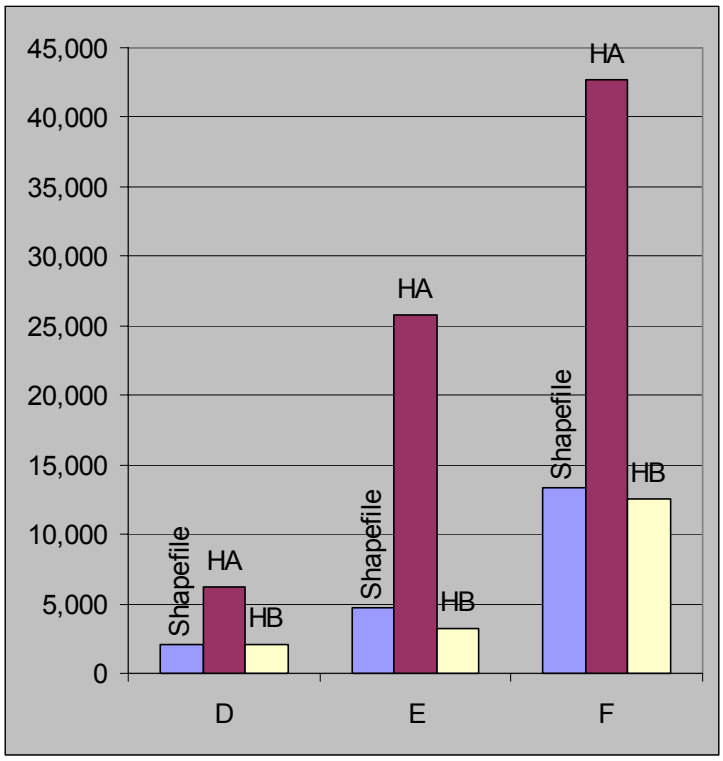
the HDF5 formats, particularly format HB, which can have a large amount of empty space, as illustrated in Figure 9.

The objects in each of the files was compressed using HDF5's compression feature. Partial results are shown in Figure 14 and Figure 15. It will be noted that for the small files, HDF5 files are still much larger, but for large files, the HB format shows comparable size with the corresponding shape files and in two cases are actually smaller. These differences illustrate the benefits of a flexible format that supports different storage structures that may be exploited to improve storage utilization. They also illustrate the value of having data compression support for individual objects.

Figure 14. File sizes resulting from application of HDF5 compression to vector data.

Shapefile name	Total # of shapes	Max. # vertices for any shape	Size (KB)		
			Shapefile	HA	HB
A	1	66	1	14	15
B	44	12	3	23	18
C	219	1,632	103	353	137
D	2,253	38,725	2,050	6,162	2,105
E	11,576	500	4,683	25,746	3,257
F	8,877	500	13,357	42,743	12,570

Figure 15. File sizes resulting application of HDF5 compression to vector data, showing only large file results.



4.4 Access studies

A second measure of applicability of a file format is the speed with which it provides access to data. Our investigations of this important dimension are currently underway, and we can only report partial results here. More will be reported on this work as we work through the next phase of the project.

4.4.1 Grid-based formats

As with storage, the regularity of grid-based formats makes it possible to implement features in a general purpose format that can greatly enhance performance. These include parallel I/O, tiling, compression, and partial access.

Although we did not do any work in Phase 1 on I/O do gridded data, we did do some work in connection with another research project[12] that has interesting implications for NARA. In this study, we are examining factors that can have an influence on I/O throughput, and in these experiments we examined the effect of compressed I/O on distributed access to data. We were particularly interested in whether compression can improve access for very large images, such as a very high resolution DOQ.

[12] "Programming Models for Scalable Parallel Computing", a DOE-sponsored project in which NCSA's role is to transfer technology from high-end computer science research to HDF5.

In these experiments, two procedures were compared:

Compressed I/O:
- compress data, then write compressed image
- read compressed image, then uncompress

Uncompressed I/O
- perform I/O without compression

Compressed I/O speed was measured against uncompressed I/O speed on two kinds of storage: a file system mounted on a local disk, and an NFS-mounted file system. The latter is a surrogate for a distributed file system in the sense that the system accesses data over a network. In both cases simple UNIX I/O operations were used.

These experiments suggest that data compression can indeed offer performance advantages, particularly when I/O is performed to a non-local file system. When writing to a local disk (Figure 16), unless the data is highly compressable (0-10% of the original size), there is little to be gained from compressing the data before writing it to disk. The real benefit comes from writing to an NFS mounted file (Figure 17). In that scenario, we see better throughput for almost every level of data compressibility. Based on these findings, we plan to implement the compressed I/O method as a feature in HDF5. Applications may invoke this method when they want to access data files reside on slow storage such as network disk or remote file servers.

Figure 16. Compressed I/O to local disk. With fast CPU speed, Compressed I/O wins when data is highly compressible. Otherwise it is better not to compress.

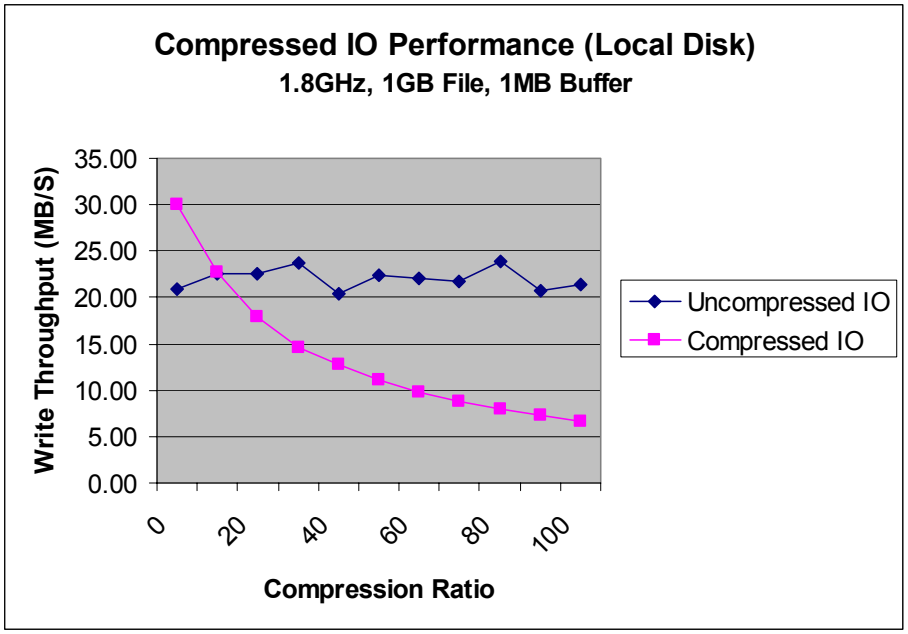
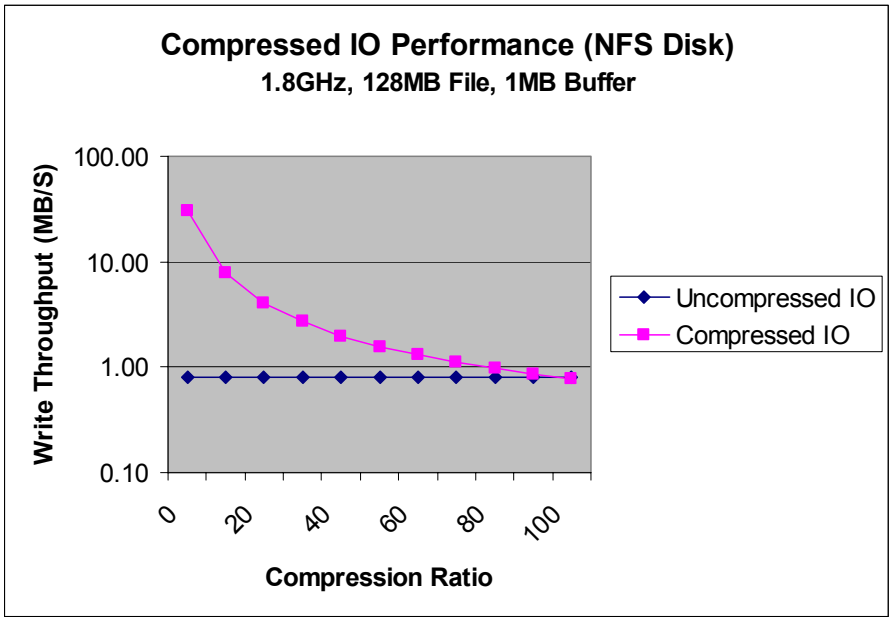


Figure 17. On a non-local (NFS mounted) file system, compression wins most of the time.

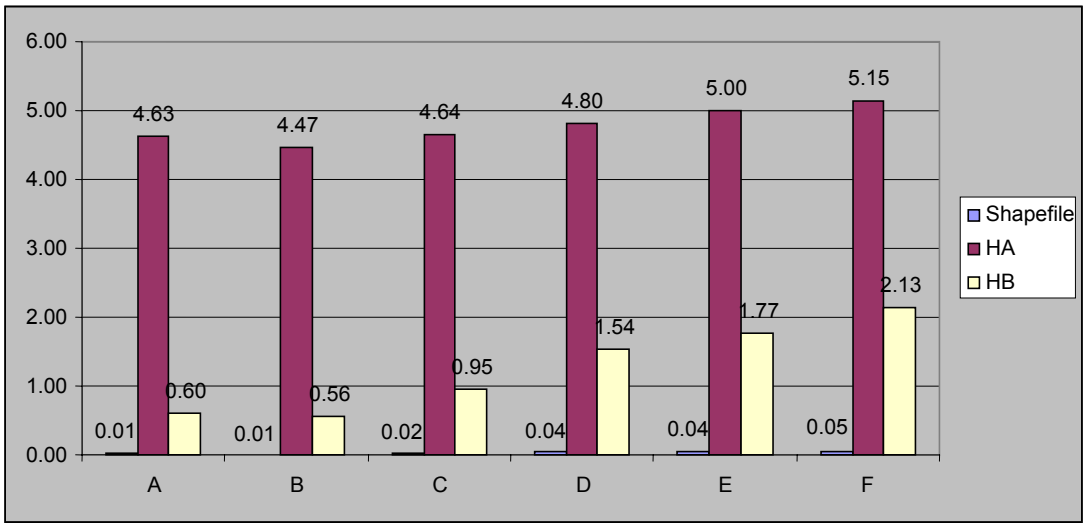


4.4.2 Vector formats

Vector access presents a genuine challenge for self-describing formats such as HDF5 that do not include vectors as native objects. Extra information must be accessed and processed in order to access a given object, and this takes considerable time. In contrast, a format such the Shapefile, which is designed specifically for fast access, has to deal with none of this extra accessing and processing.

We have carried out a number of experiments to see whether the Shapefile organization does indeed offer performance advantages, comparing Shapefile access speed with HDF5. The different organizational structures described in section 4.3.2 were compared, as well as several others. These investigations reveal that there is indeed a considerable performance gain to be had by using the Shapefile format for vector data access. One chart (Figure 18) will suffice to illustrate difference.

Figure 18. Total access time for 1000 random accesses to all six sample Shapefiles and corresponding HDF5 files.



In Phase 2, we will explore further the reasons for these dramatic differences, but in the meantime this outstanding performance for Shapefiles illustrates the advantages to be gained from the use of a specialized format.

5 Summary and conclusions

This pilot study in cross-disciplinary technology transfer has begun an examination of ways that scientific data management technologies might help address challenges that NARA and other federal archives can face in dealing with federal collections of digital geospatial data.

We examined a sampling of geospatial datatypes, including 2D grids, multi-layer 2D grids, 3D volumes, the swath, and vector types. We identified characteristics of these geospatial types that match well with data types from the general scientific community, and postulated that scientific data formats such as HDF5 might provide a good format and software system for preservation of certain geospatial data.

Mappings were developed between sample geospatial datasets and the HDF5 or HDF-EOS 5 formats, and sample data sets were converted from their original formats to HDF5 or HDF-EOS 5. The object types, data types, and structures provided in HDF5 and HDF-EOS 5 were able quite adequately to accommodate all of the information that was stored in the original geospatial formats.

We then explored some of the implications of these conversions in terms of file size and, to a lesser extent, object access speed. It was found that grid types not only mapped well to HDF5 and HDF-EOS 5, but they often also required less space, sometimes considerably less. Access speed was not studied for grids, but is planned for a second phase of this research.

The results for vector types were more mixed. Vector types mapped well to HDF5 datatypes, but required much more disk space than the original format. Alternate, less self-descriptive mappings from Shapefiles to HDF5 did result in good space utilization

when HDF5's compression feature was invoked. Initial investigations of I/O performance were also disappointing, with Shapefile access outperforming HDF5 substantially on random access to large numbers of features. This investigation will continue in the second phase of the research.

6 Acknowledgements

This report is based upon work supported primarily by the Electronic Records Archive of the US National Archives and Records Administration under grant number NARA NSF 02-02GPG. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Archives and Records Administration.

Additional support was provided by a Cooperative Agreement with NASA under NASA grant NAG 5-2040 and NAG NCCS-599, and by the Department of Energy under the grant DOE DEFC02-01ER 25508. Other support provided by NCSA and other sponsors and agencies (See: http://hdf.ncsa.uiuc.edu/acknowledge.html).

7 References

1. Baldwin, Chuck, Ghaleb Abdulla and Terence Critchlow, "Multi-resolution modeling of large scale scientific simulation data", *Proceedings of the twelfth international conference on Information and knowledge management*, 2003, pp. 40-48.

2. Barrodale Computing Services Ltd. (BCS), "Storing and Manipulating Gridded Data in Databases". 2002. http://www.barrodale.com/grid_Demo/gridInfo.pdf.

3. ESRI. "ESRI Shapefile Technical Description. An ESRI White Paper". Environmental Systems Research Institute, Inc. July 1998. http://www.esri.com/library/whitepapers/pdfs/shapefile.pdf.

4. The HEASARC (High Energy Astrophysics Science Archive Research Center), "FITS – The Astronomical Image and Table Format." http://fits.gsfc.nasa.gov/fits_home.html.

5. Klein, Larry. "HDF-EOS Interface Based on HDF5, Volume 1: Overview and Examples." Technical Paper 175-EMD-001, October 2003. http://hdfeos.gsfc.nasa.gov/hdfeos/Info/Info_docs/HDF-EOS/guides/HDFEOS5.1.6_ug_vol1_oct_2003.pdf .

6. NCSA, "The NCSA HDF Home Page." http://hdf.ncsa.uiuc.edu/.

7. NCSA, "HDF5 – A New Generation of HDF." http://hdf.ncsa.uiuc.edu/HDF5/.

8. Taaheri, E.M. and David Wynne. "An HDF-EOS and Data Formatting Primer for the ECS Project." White Paper 175-WP-001-002, March 2001. http://hdfeos.gsfc.nasa.gov/hdfeos/Info/Info_docs/HDF-EOS/guides/HDFEOS_Primer_mar_2001.pdf

9. Unidata, "NetCDF". http://www.unidata.ucar.edu/packages/netcdf/.