Scientific dataformats

Seminar thesis

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

Author:	Sebastian Döbel
E-mail:	3doebel@informatik.uni-hamburg.de
Student number:	6542135
Study programme:	Computing in Science, Schwerpunkt Physik
Supervisor:	Konstantinos Chasapis

Dresden, 10.11.2016

Abstract

In this work I will introduce some spescientific data formats from different sciences as well as the more common format HDF5.

Contents

1 Requirements

If you want to create a new data format or choose one of the existing ones for your application you have to think of the following requirements:

- filesize
- convertibility to other known formats
- archivability, backwards compatibility
- portability
- interchangeability
- searchability
- human readablity (text based)

One of the most important requirements is the file size. The size of the produced file is one important indicator of the further cost of your (memory intensive) application because the file size implies storage size and therefore you need hardware.
For that reason, you have to think of which data needs to be stored in your file(s) and which you better recalculated in your application.
Moreover, file size is strongly connected with other aspects such as compression, precision of your values and additional meta data.

Additional meta data is also strongly connected to the other previously named requirements.

Furthermore, you have to mention portability which means that your file will be equally represented on different platform, e.g. line endings for textfiles are represented differently on unix systems (\n) and on Microsoft Windows systems (\r\n) or representation of integer values (32-bit and 64-bit). This problems are often solved by self-containing data formats which contain meta data for an exact description of data typs.
But portability must be also mentioned in application context.

Another requirement is archivability which goes on with backwards compatibility. Several data formats are designed for it e.g. FITS (section 3.2.

Nevertheless, the major point is what are already established formats in you application context and if you design a new data format is your format convertible to them. Nobody is happy to use a format which is not possible to use with other applications.
First come, first served, is reality even on data formats.

2 Mathematical objects

Here I am going to give an introduction of how typical mathematical objects could be transformed to storable data.

2.1 Graphs

Graphs like figure 2.1 are often put into a second place but there are many usages of them.
The most known usage of them are models traffic systems like the hvv map which can be seen in figure 2.2. Every station can be interpreted as an vertex (node) and connections between them as edges. But graphs are also very important in modern physics. State transitions of excited electrons are only one example taken from physics.

I will only give some basic ideas how graphs can be stored with the help of adjacency lists and matrices. In addition to it, there are incidence matrices which are similar to adjacency once. For advanced graph representation one should look into graph databases e.g. Neo4j[1].

[1] `https://neo4j.com/`

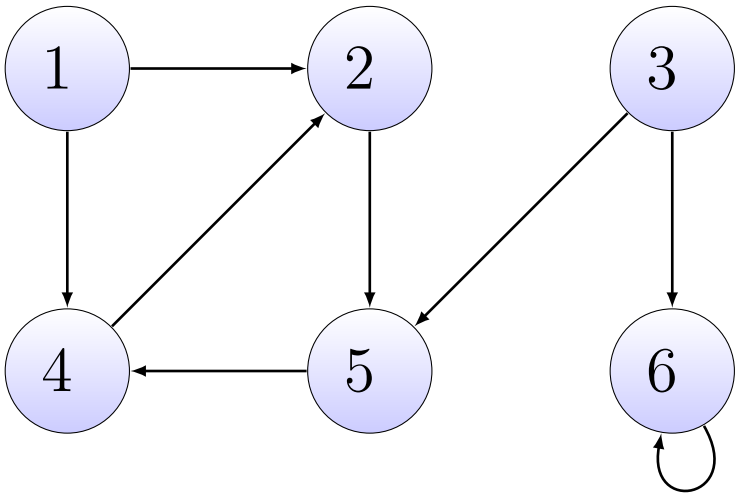


Figure 2.1: directed graph

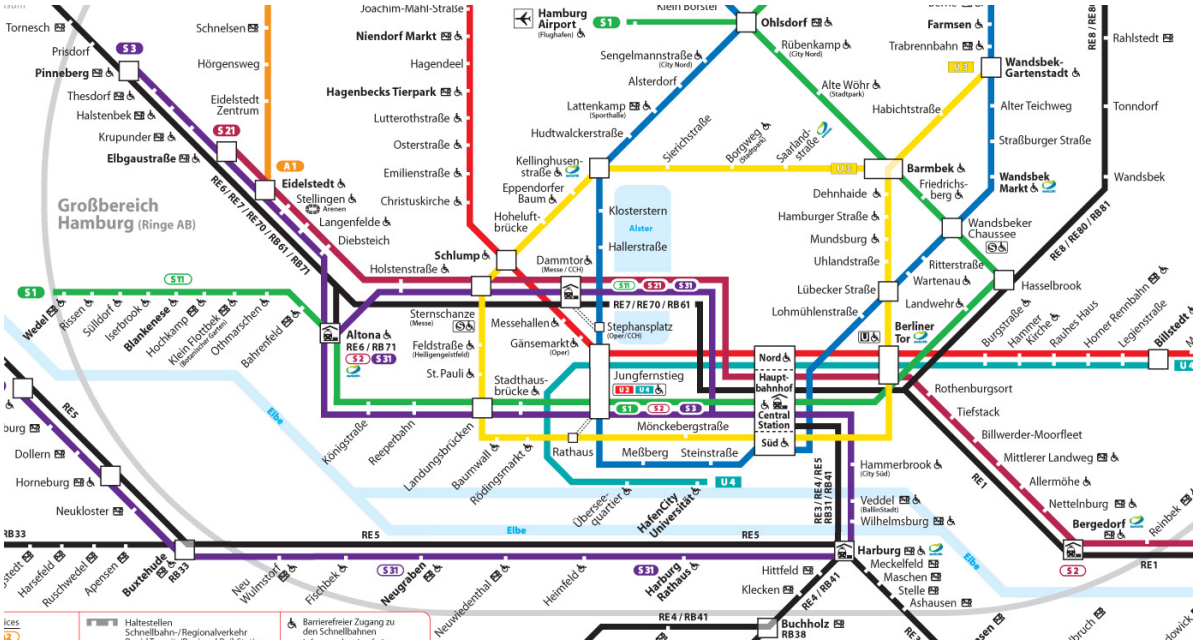


Figure 2.2: extract from the hvv map (15)

2.1.1 Adjacency list

One possibility to convert a graph to the text are adjacency list.
The basic idea is to choose one vertex and put the neighbour of every outgoing edge into a list. Then go on with the next vertex. If you have weighted graphs, just put a tuple of vertex and weight into the list.
One should mention that these list are usually unsorted.

By doing this you get a list of size $\Theta(|V| + |E|)$. Therefore, it is reasonable to use this only for sparse graphs ($|E| \ll |V|^2$).(3, S. 531)

An adjacency list for figure 2.1 could be:

$1 : \{2, 4\}$
$2 : \{5\}$
$3 : \{6, 5\}$
$4 : \{2\}$
$5 : \{4\}$
$6 : \{6\}$

2.1.2 Adjacency matrix

In addition to adjacency lists one can also use matrices to store graphs. This is useful for nearly full graphs ($|E| \sim |V|^2$).

The elements $a_{i,j}$ of an matrix $A^{|V|\times|V|}$ are build by:

$$a_{i,j} = \begin{cases} 1 & \text{if } (i,j) \in E \\ 0 & \text{else} \end{cases} \tag{2.1}$$

If you have weighted graphs just replace 1 by the weight of the edge.

By using matrices instead of lists you have faster access to specific edges and so you can faster check if 2 vertexes are connected. But on the other hand you have a constant size of $\Theta(|V|^2)$. (3, S. 531)

An adjacency list for figure 2.1 could be:

$$\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccccc} 1 & 2 & 3 & 4 & 5 & 6 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

2.2 Matrices

Some other often used data structure are grids and matrices.

2.2.1 Grids and common matrices

In climate science you often have to deal with grids because you have to represent earth's surface and atmosphere. Therefore you often have to deal with 2d (e.g. surfaces) or 3d (e.g. atmosphere) grids.

Each point in a grid is represented by coordinates (e.g. Cartesian coordinates) or length and angle (polar/spheric coordinates) which can be put into a (multidimensional) table. This tables can be interpreted as matrices and table legends can be interpreted as vectors.

To save matrices just write them element and column/row-wise into a file. You separate elements and rows by special chars or use fixed lengths.

2.2.2 Sparse matrices

Sparse matrices are matrices where many elements are 0. They often occur as results by discretizing partial differential equation (e.g. heat equation, Maxwell equation, Schrödinger equation) but also as results of adjacency matrices (see 2.1.2). They are usually wanted because there are very efficient algorithms to do calculation with them (e.g. Thomas algorithm for system of linear equations).

To save them you only have to save non-0 elements with their position or if you have band structures just save start indices.

$$\begin{pmatrix} a_{11} & a_{12} & 0 & \cdots & \cdots & \cdots & \cdots & 0 \\ a_{21} & a_{22} & a_{23} & \ddots & & & & \vdots \\ 0 & a_{32} & a_{33} & a_{34} & \ddots & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \ddots & \ddots & & \vdots \\ \vdots & & \ddots & \ddots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & \ddots & a_{(n-2)(n-3)} & a_{(n-2)(n-2)} & a_{(n-2)(n-1)} & 0 \\ \vdots & & & & \ddots & a_{(n-1)(n-2)} & a_{(n-1)(n-1)} & a_{(n-1)n} \\ 0 & \cdots & \cdots & \cdots & \cdots & 0 & a_{n(n-1)} & a_{nn} \end{pmatrix}$$

Figure 2.3: band matrix, especially tridiagonal matrix

3 Examples of scientific data formats

Nearly every science has it own established format. In the following, I will introduce some of them.

3.1 Computational life science - FASTA

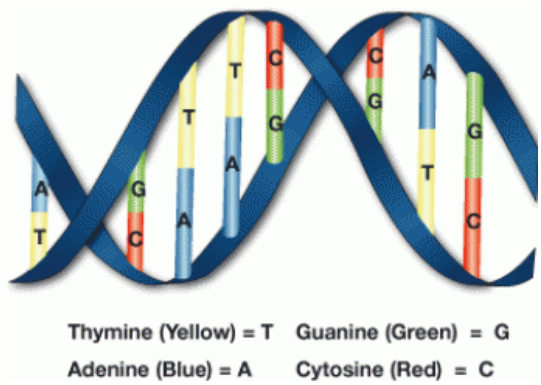


Figure 3.1: dna sequnce (11)

In computational life science you often have to deal with primary structures of nucleic acids or proteins. One of the most used format to deal with them is FASTA.

FASTA is usually used within the FASTA or BLAST algorithm. This are algorithms to compare different sequences including searches of equal or similar sub sequences.

FASTA is a purely ASCII text based format where every acid get a letter (see table 3.1). Here the user has to pay attention because every letter can be interpreted as nucleic or as amino acid. So, it is not possible to say if AAGAGGA is a sequence of adenine and guanine or alanine and glycine.
To make it out, the headline can give you a hint. Every FASTA sequence starts with "<" followed by an optional unique name and description.
Unfortunately there is no standard how this unique name and description will be created. Therefore, there are conflicts between the big databases.
An example FASTA file can be seen in listing 3.1.

Moreover it is possible to put multiple sequences into one file. Each sequences starts with a new "<".
There is also some variation FASTQ where acids' quality scores are encoded into the letter.

Code	Meaning
A	**A**denine
G	**G**uanine
T	**T**hymine
K	K = GT (**K**etone)
-	gap of indeterminate length

(a) extract of nucleic acid codes

Code	Meaning
A	alanine
G	glycine
T	threonine
K	lysine
X	any
*	translation stop
-	gap of indeterminate length

(b) extract of amino acid codes, 25 acids + 3 special codes

Table 3.1: FASTA translation codes (20)

DNA as well as protein sequences are very huge files. Therefore compression is a big topic when talking about FASTA. General purpose tools like gzip fall short on it. Therefore there is a huge variety of compression algorithms. One of the most used is *MFCompress*. In comparison to gzip you can get about 50% additional compression and on highly redundant datasets up to 8-fold of gzip. However, you get larger computation time. (16)

As already mentioned FASTA has a huge distribution. Therefore there are a lot of tools using FASTA. Some of them like *Genome Tools & libgenometools* and *Vmatch* are developed in collaboration of the ZBH in the group of Prof. Dr. Stefan Kurtz[1].
Other applications using FASTA are e.g. *SeqAn* and *FASTA/FASTQ parser in C*[2]. But there are also a lot of ruby, python and perl scripts dealing with FASTA data.
Moreover most of the huge databases for DNA and protein sequences such as National Centre for Biotechnology Information (NCBI) and European Bioinformatics Institute (EBI) offer FASTA data.

[1] `http://www.zbh.uni-hamburg.de/kurtz`
[2] `http://lh3lh3.users.sourceforge.net/parsefastq.shtml`

```
1 >gi|2196610|emb|CAB09441.1| cytochrome b, partial (mitochondrion)
    ↪ [Acomys spinosissimus]
2 MTNIRKTHPLLKIINHAFIDLPAPSNITSWWNFGSLLGICLIIQIITGLFLAMHYTSDTSTAFSSVTHIC
3 RDVNYGWLIRYLHANGASMFFICLFMHVGRGIYYGSYTYMETSNIGIILLFAVMATAFMGYVLPWGQMSF
4 WGATVITNLLSAIPYIGTNLVEWIWGGFSVDKATLTRFFAFHFILPFIIAALAMVHLLFLHETGSNNPTG
5 INSDSDKIPFHPYYTMKDLLGAFILLLTLLALVLFSPDLLGDPDNYTPANPLNTPPHIKPQWYFLFAYAI
6 LRSIPNKLGGVLALVLSILVLAILPLIHTSKQRSLMFRPISQTLFWILVANLLILTWIGGQPVEHPFIII
7 GQLASISYFTIILILIPISGLIENKMMKWN
```

Listing (3.1) cytochrome b of Southern African spiny mouse (Acomys spinosissimus) (german: Zwergstachelmaus) (figure 3.2a); taken from (1)

(a) Southern African spiny mouse (Acomys spinosissimus) (4)

Figure 3.2: Southern African spiny mouse (Acomys spinosissimus)

3.2 Astronomy - FITS

The Flexible Image Transport System (FITS), standardized 1981, is the most widely used data format within astronomy. It's designed to store datasets consisting multidimensional arrays (images) and 2d tables. Furthermore it is designed for long-term archival storage and so backwards compatibility. The motto is *"Once FITS, always FITS"*. As a result FITS is also used in the Vatican Apostolic Library.

A FITS file contains at least one header-data-unit (HDU). The first HDU called *"primary HDU"* contains an 1 to 999 dimensional array of integers or floats and typically represents an 1-3d image. Additional HDUs are called *"extensions"*. Their data units could be images, ASCII or binary tables.

The "Header Unit" of each HDU has strict style guidelines. They are ASCII formatted containing key-value-pairs. Each pair has to have a length of 80 chars. Key names are only allowed to be up to 8 characters containing upper case letters, digits, the hyphen and underscore character. Values can be integer, floats, complex values, character strings (enclosed in single quotations) or boolean (character T or F). Comments are introduced by the slash character.

A primary header of a 2d image containing 100×100 pixel with 8bit looks like:

```
SIMPLE␣␣=␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣T␣/␣file␣conforms␣to␣FITS␣
    ↪ standard␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.
BITPIX␣␣=␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣8␣/␣number␣of␣bits␣per␣data␣
    ↪ pixel␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.
NAXIS␣␣␣=␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣2␣/␣number␣of␣data␣axes␣␣␣␣␣␣␣␣
    ↪ ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.
NAXIS1␣␣=␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣100␣/␣length␣of␣data␣axis␣1␣␣␣␣␣␣
    ↪ ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.
NAXIS2␣␣=␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣100␣/␣length␣of␣data␣axis␣2␣␣␣␣␣␣
    ↪ ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.
[␣optional␣custom␣keywords␣like␣ORIGIN,␣DATE,␣TIME,␣
    ↪ FILTER1,...␣]␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.
END␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
    ↪ ␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣.
```

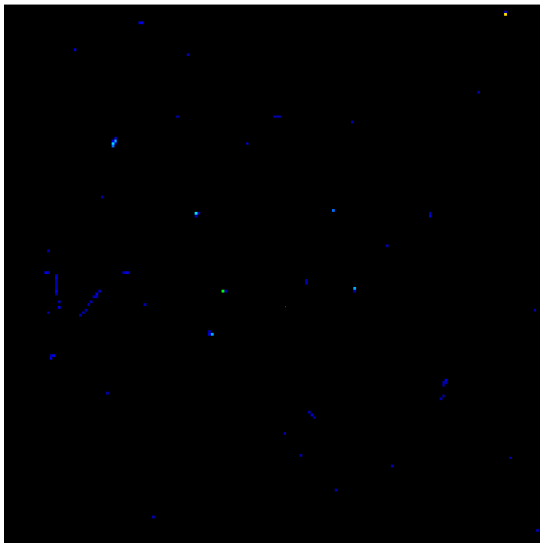
Listing 3.2: FITS example header. (the dot at every line end has to be ignored)

Additionally, the size of every header and data unit has to be a multiple of 2880byes (36 times 80-byte), if its not, it has to be padded by blank records. Thus, the data unit is optional. HDUs only consisting header are valid. (14)

Despite this benefits of good backwards compatibility, portability and customization for astronomic data, FITS is a bit criticized. New features are often not integrated into FITS because they would break backwards compatibility to older version. As a result, fast and especially parallel writing of huge datasets a big problem. One solution for this problem is to use HDF5 (section 4) which is convertible bijectively. (13, 9, 17, 19)

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    3 / number of data axes
NAXIS1  =                  200 / length of data axis 1
NAXIS2  =                  200 / length of data axis 2
NAXIS3  =                    4 / length of data axis 3
EXTEND  =                    T / FITS dataset may contain extensions
COMMENT   FITS (Flexible Image Transport System) format is defined in 'Astronomy
COMMENT   and Astrophysics', volume 376, page 359; bibcode: 2001A&A...376..359H
BSCALE  =                1.0E0 / REAL = TAPE*BSCALE + BZERO
BZERO   =                0.0E0 /
OPSIZE  =                 2112 / PSIZE of original image
ORIGIN  = 'STScI-STSDAS'       / Fitsio version 21-Feb-1996
FITSDATE= '2004-01-09'         / Date FITS file was created
FILENAME= 'u5780205r_cvt.c0h'  / Original filename
ALLG-MAX=           3.777701E3 / Data max in all groups
ALLG-MIN=          -7.319537E1 / Data min in all groups
ODATTYPE= 'FLOATING'           / Original datatype: Single precision real
[....]
              / SCIENCE INSTRUMENT CONFIGURATION

MODE    = 'FULL'               / instr. mode: FULL (full res.), AREA (area int.)
SERIALS = 'OFF'                / serial clocks: ON, OFF

              / IMAGE TYPE CHARACTERISTICS

IMAGETYP= 'EXT                 ' / DARK/BIAS/IFLAT/UFLAT/VFLAT/KSPOT/EXT/ECAL
CDBSFILE= 'NO                  ' / GENERIC/BIAS/DARK/PREF/FLAT/MASK/ATOD/NO
PKTFMT  =                   96 / packet format code

              / FILTER CONFIGURATION

FILTNAM1= 'FR533P15'           / first filter name
FILTNAM2= '         '          / second filter name
FILTER1 =                   69 / first filter number (0-48)
FILTER2 =                    0 / second filter number (0-48)
FILTROT =                 15.0 / partial filter rotation angle (degrees)
LRFWAVE =          4877.000000 / linear ramp filter wavelength
[....]
```

Listing (3.3) extract from the header of figure 3.3a



(a) image

Figure 3.3: example picture taken on HST WFPC II (12)

3.3 Climate science - GRIB

The GRidded Binary data format (version 1) or General Regularly-distributed Information in Binary form (since version 2) standardized by the World Meteorological Organization's (WMO) Commission for Basic Systems (CBS) is very popular format in climate science but also for private and commercial usage of weather data. Examples for non-scientific usage are shipping, hobby sailing, surfing or trekking, climbing.
It's designed to store weather and climate indicators such as temperature and rainfall but also not typically mentioned data e.g. wave height.

Grid consists of binary 2d matrices with discretization of the room. It contains a collection of self-contained, independent (no references to other records) records (messages). Each record consists a header and the data. The data is saved in integers and needed to be scaled in version 1. Moreover in version 2 compression of these data was added.

Another benefit in the GRIB structures are inventory. They can be created by the user and act as an "table of content" with users meta data and position (offset) of connected data. As a result, it is possible to only transfer needed data parts which is quite useful for huge data on low networks like USW or satellite.

Some tools for working GRIB are:

- *wgrib* like typical UNIX filter
- *degrib* creates indizes for faster access
- *dkrz_readgrib*[3]
- GUI: GRIBview[4]

As already mentioned, GRIB is not only used in scientific applications, like post-processing package PINGO developed by the DKRZ, but also in more common applications. One of them is zyGrib. zyGrib offers a Tool to visualize GRIB-data but also provides a server with periodically generated data. Moreover there is the website `PassageWeather.com` offering GRIB data.
Besides, there are even smart phone applications like Marine Weather | Sail Grib Free dealing with weather data provided in GRIB. (18)

3.4 Climate & geo science - NetCDF

In climate science as well as in geo science there is also the Network Common Data Format (NetCDF). NetCDF based originally on NASA's Common Data Format but they are not compatible anymore. It is an open standard maintained by University Cooperation for Atmospheric Research (UCAR) now.

[3] `http://mms.dkrz.de/pdf/klimadaten/static/Pingo/post/post.dkrzgrib.html`
[4] `http://www.theyr.com`

NetCDF stores data in self-contained, platform independent datasets into a binary format. In the background NetCDF4 uses HDF5, which will be discussed later in section 4.6.

NetCDF consists 4 basic objects.
The first object called “dimensions“ is for measurands (scale) like time, length,.... Every dimension has name and size. Only one size can be unlimited (dataset dimension).
Moreover datasets consist “variables“ where an array of measured data (every value has to have the same size) is put into. They also have a name, a datatype and they can have a shape. One dimensional variables with the same name as their only dimension are called “coordinate variables“.
Another object are “attributes“ where users can store any additional information for the dataset (meta data). They can be used in variables or global. (2)

4 HDF5

4.1 Introduction

The Hierarchical Data Format (HDF) standardized 1988 by National Centre for Supercomputing Applications (NCSA) is now developed by HDF Group.

It's designed with backwards compatibility and a huge platform support. At the moment official support exists for C, C++, Fortran and Java. But there is also a huge third-party support for Go, Python, R, MATLAB (Scilab, Octave), Mathematica, ERLANG, Perl, LabVIEW, ... which are usually wrapper of the C functions.

HDF5 files are binary files containing 4 objects. With groups and datasets one can build a structure similar to a filesystem containing folders and files. Groups have a name and additional attributes. They are able to contain other groups and datasets. A dataset can be a single value or a table with any dimension having a name and additional optional attributes.
These additional attributes contain any information ("meta data") given from users for users e.g. simulation parameter, author, ...
Furthermore, there is an object called meta data, not to mix up with attributes, representing information about content e.g. specific size of datatypes (api).

4.2 Modules

HDF5 is organized in separate but not independent modules. A list of them is shown in table 4.1.

API	Description
H5	Library Functions: general-purpose H5 functions
H5A	Annotation Interface: attribute access and manipulation routines
H5D	Dataset Interface: dataset access and manipulation routines
H5E	Error Interface: error handling routines
H5F	File Interface: file access routines
H5G	Group Interface: group creation and operation routines
H5I	Identifier Interface: identifier routines
H5L	Link Interface: link routines
H5O	Object Interface: object routines
H5P	Property List Interface: object property list manipulation routines
H5R	Reference Interface: reference routines
H5S	Dataspace Interface: dataspace definition and access routines
H5T	Datatype Interface: datatype creation and manipulation routines
H5Z	Compression Interface: compression routine(s)

Table 4.1: List of HDF5 modules (6)

4.3 Datatypes

HDF5 also offers a complex datatype system which has hierarchical order (see figure 4.1). To get it from the base:
HDF5 offers atomic types which can't be decomposed into smaller units. Atomic types are:

- integer
- float
- string
- date
- time
- bitfield
- reference
- opaque

Moreover, HDF5 offers the possibility to compose these types. The possible ways of composition are called "composite". Composite types are:

- array
- variable length

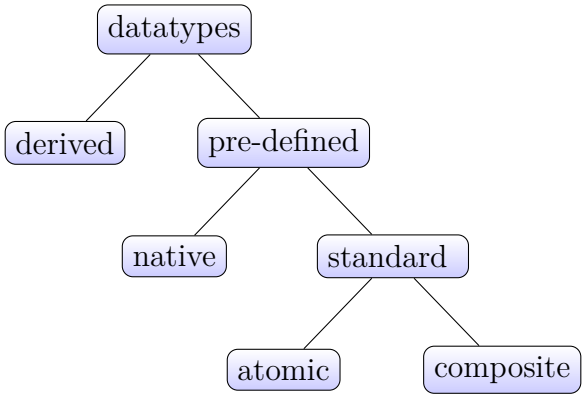


Figure 4.1: hierarchy of datatypes in HDF5

- enumeration
- compound datatypes

Atomic and decomposite types are the same on every machine to ensure portability. Therefore, HDF5 has it own implementation for languages' standard types. Besides this standard types. There are also so called "native" types. They are mostly aliases to specific standard types but they to not guarantee portability. For example *H5T_NATIVE_LONG* can be 32bit or 64bit on different systems. Moreover, on Intel based system integer types are mapped to **LE* types but on MIPS systems to **BE* types. But it also depends on the compiler.
Besides this pre-defined type it is also possible to create own derived types of them.

HDF5 offers a lot of pre-defined datatypes. But it is also possible to create own derived types of them to build complex data structures or just change size, precision or binary representation and interpretation. By changing only the binary representation and interpretation one is able to convert American date format (YYYY-DD-MM) to European format (DD-MM-YYYY).

Moreover, HDF5 offers two additional modules with extended datatypes: table and packet table. The difference between them is, that packet table is allowed to have different type sized and you are allowed to store strings with different length. In table one have a unique number of types and a unique size of types. (5)

4.4 Optimization

HDF5 offers the possibility to override system methods like cache usage to increase performance. Moreover HDF5 offers compression algorithm. There are 2 predefined standard lossless algorithms: gzip and szip. Additionally, the user is able to implement his own compression module which could be individually used for specific data. Using compression often is a struggle of size vs. performance.

Another important optimization is parallel I/O. HDF5 offers parallel I/O support for C and Fortran. C++ has only experimental support which worked in my tests.

It offers the possibility to read and write parallel into the datasets by multiple processes. In the background, HDF5 uses MPI to realize parallel I/O. Therefore, you have to have the benefits and disadvantages of MPI I/O. e.g. a parallel filesystem is needed.
Files created by parallel HDF5 are compatible with serial once. This was one main goal while developing it.

To create a file within this parallel environment you just need to create an MPI environment and change HDF5's access properties to use MPI I/O instead of standard single POSIX. (10, 7)

4.5 Tools

HDF5 offers a huge variaty of tools for post processing of already written data. To name some of them (8):

- converting HD4 to HD5 (*h4toh5*, (*h5toh4*))
- import & export e.g.
  - export dataset of an image to GIF (*h52gif*)
  - import GIF file into HDF5 file (*gif2h5*)
  - create printable version (*h5dump*)
- split & merge files (*h5repart*) (This could be used if your Filesystem limits the maximal file size e.g. FAT32 only supports 4GiB)
- copy and compression (*h5repack*)
- compare two files (*h5diff*, *ph5diff*)
- performance tests (*h5perf*, *h5perf_serial*)
- GUI: *HDFView* (Java-binding)

4.6

NetCDF used HDF5 in the background and HDF5 provides universal data structures which could be used in a variety of sciences. Therefore one could ask, why to use other formats instead of using HDF5 everywhere.

The answer is easy. By offering a lot of features and flexibility HDF5 gets rather complex which produces overhead in files but also programme code and even requires additional knowledge of users which are mostly not computer scientist. Most of the features are not used by one application. Therefore NetCDF which also provides a limited access to HDF5 function and features is still popular because it is easier to work with it.

An other aspect is that most of the formats are quite old and established in their context. Therefore, there exists good known tools in their context to work with them. As a result you can use your data files with more common applications and if you build an application you want other users to work with you have to make it compatible to standards.

5 Summary

Nearly every science has their own data format. These different formats provide sometimes same functionality but often are customised for a certain environment. Therefore, migration from one to another format are often not easy.

HDF5 is a more common data format with a huge variety of features and function but also with a huge complexity. One great benefit of HDF5 is parallel I/O by using MPI I/O.

Bibliography

[1] *cytochrome b of Southern African spiny mouse (Acomys spinosissimus) in FASTA format*. http://www.ncbi.nlm.nih.gov/protein/2196610?report=fasta. – Last visited 28.04.2016

[2] *Overview on netCDF-Files*. Online: http://desktop.arcgis.com/de/arcmap/10.3/manage-data/netcdf/a-quick-tour-of-netcdf-data.htm. – Last visited 09.05.2016

[3] Cormen, Thomas H. ; Leiserson, Charles E. ; Rivest, Ronald ; Stein, Clifford ; Molitor, Paul: *Algorithmen - Eine Einführung*. Bd. 2. Oldenburg : Oldenbourg Wissenschaftsverlag GmbH, 2007

[4] Family Tree DNA: *Picture of DNA sequence*. Online: (https://www.familytreedna.com/img/understanding-dna/dna_diagram.gif). – Last visited 02.11.2016

[5] HDF5 group: *HDF5 Datatypes*. Online: https://support.hdfgroup.org/HDF5/Tutor/datatypes.html. – Last visited 02.08.2016

[6] HDF5 group: *HDF5 Modules*. Online: https://support.hdfgroup.org/HDF5/Tutor/api.html. – Last visited 02.08.2016

[7] HDF5 group: *Introduction to HDF5 by HDF Group*. Online: https://www.hdfgroup.org/HDF5/doc/H5.intro.html. – Last visited 09.05.2016

[8] HDF5 group: *Software using HDF5*. Online: https://www.hdfgroup.org/tools5desc.html. – Last visited 09.05.2016

[9] Kayser, Rainer: Astro-Dateiformat für Vatikanische Bibliothek. In: *astronews.com* (2012), January

[10] Kirchhart, Lukas: *Hierarchical Data Format vs. Textbasierte Datenformate*. Aachen, RWTH Aachen, seminar paper, December 2009

[11] Motycka, Vladimir: *Picture of southern african spiny mouse*. Online: (http://sciencepole.com/southern-african-spiny-mouse/). – Last visited 02.11.2016

[12] NASA/GSFC: *Fits example image*. Online: http://fits.gsfc.nasa.gov/samples/WFPC2u5780205r_c0fx.fits. – Last visited 2.11.2016

[13] NASA/GSFC: *FITS Website*. Online: `http://fits.gsfc.nasa.gov/`. – Last visited 28.04.2016

[14] NASA/GSFC: *A Primer on the FITS Data Format*. Online: `http://fits.gsfc.nasa.gov/fits_primer.html`. – Last visited 28.04.2016

[15] NBE NORDBAHN EISENBAHNGESELLSCHAFT MBH & CO. KG: *HVV map*. Online (Online: `http://www.nordbahn.de/reiseplanung/uebersichtskarten.html`). – URL `http://www.nordbahn.de/reiseplanung/uebersichtskarten.html`. – 02.11.2016

[16] PINHO, Armando J. ; PRATAS, Diogo: MFCompress: a compression tool for FASTA and multi-FASTA data. In: *BIOINFORMATICS APPLICATIONS NOTE* (2013)

[17] PRICE, D. C. ; BARSDELL, B. R. ; GREENHILL, L. J.: HDFITS: porting the FITS data model to HDF5. In: *Astronomy and Computing* 12 (2015), October, S. 212–220

[18] SCHERER, Roman: *Konzeption und Implementierung einer Community-Plattform für Surfer*. Berlin, Humboldt-Universität zu Berlin, diploma thesis, 2009

[19] SCHWARZBURG, Stefan: *Eine Software zur Echtzeitanalyse von experimentellen Daten im Flexible Image Transport System (FITS)*, Eberhard Karls Universität Tübingen, diploma thesis, June 2005

[20] TAO, Tao: *FASTA letter codes*. Online: `http://www.ncbi.nlm.nih.gov/staff/tao/tools/tool_lettercode.html`. – Last visited 09.05.2016