

HDF5 Advanced Topics

Datatypes

HDF and HDF-EOS Workshop IX

November 30, 2005

Goal

Introduce HDF5 datatypes

Topics

- Overview of HDF5 datatypes
- Simple atomic datatypes
- Composite atomic datatypes
- Compound datatypes
- Discovering HDF5 datatype

Overview of HDF5 Datatypes

Datatypes

- A datatype is
 - A classification specifying the interpretation of a data element
 - Specifies for a given data element
 - the set of possible values it can have
 - the operations that can be performed
 - how the values of that type are stored
 - May be shared between different datasets in one file

General Operations on HDF5 Datatypes

- Create
 - `H5Tcreate` creates a datatype of the `H5T_COMPOUND`, `H5T_OPAQUE`, and `H5T_ENUM` classes
- Copy
 - `H5Tcopy` creates another instance of the datatype; can be applied to any datatypes
- Commit
 - `H5Tcommit` creates an Datatype Object in the HDF5 file; comitted datatype can be shared between different datasets
- Open
 - `H5Topen` opens the datatypes stored in the file
- Close
 - `H5Tclose` closes datatype object

Programming model for HDF5 Datatypes

- Create
 - Use predefined HDF5 types
 - Create compound or composite datatypes
 - Create a datatype (by copying existing one or by creating from the one of H5T_COMPOUND(ENUM,OPAQUE) classes)
 - Create a datatype by queering datatype of a dataset
 - Open committed datatype from the file
 - Set datatype properties (length, precision, etc.)
- (*Optional*) Discover datatype properties (size, precision, members, etc.)
- Use datatype to create a dataset/attribute, to write/read dataset/attribute, to set fill value
- (*Optional*) Save datatype in the file
- Close
 - No need to close for predefined datatypes

Simple Atomic HDF5 Datatypes

HDF5 Atomic Datatypes

- Atomic types classes
 - standard integers & floats
 - strings (fixed and variable size)
 - pointers - references to objects/dataset regions
 - enumeration - names mapped to integers
 - opaque
 - bitfield
- Element of an atomic datatype is a smallest possible unit for HDF5 I/O operation
 - Cannot write or read just mantissa or exponent fields for floats or sign field for integers

HDF5 Predefined Datatypes

- HDF5 Library provides predefined datatypes (symbols) for all atomic classes except opaque
 - `H5T_<arch>_<base>`
 - Examples:
 - `H5T_IEEE_F64LE`
 - `H5T_STD_I32BE`
 - `H5T_C_S1`
 - `H5T_STD_REF_OBJ`, `H5T_STD_REF_DSETREG`
 - `H5T_NATIVE_INT`
- *Predefined datatypes do not have constant values; initialized when library is initialized*

HDF5 Predefined Datatypes

- Operations prohibited
 - Create (H5Tcreate)
 - Close (H5Tclose)
- Operations permitted
 - Copy (H5Tcopy)
 - Set/get size and other properties

When to use HDF5 Predefined Datatypes?

- In datasets and attributes creation operations
 - Argument to `H5Dcreate` or to `H5Acreate`
- In datasets and attributes read/write operations
 - Argument to `H5Dwrite/read`, `H5Awrite/read`
 - Use **`H5T_NATIVE_*` types for application portability**
- To create user-defined types
 - Fixed and variable-length strings
 - User-defined integers and floats (13-bit integer or non-standard floating-point)
- In composite types definitions
- *Do not use for declaring variables*

Storing strings in HDF5

- Array of characters
 - Access to each character
 - Extra work to access and interpret each string
- Fixed length

```
string_id = H5Tcopy(H5T_C_S1);
H5Tset_size(string_id, size);
```

 - Overhead for short strings
 - Can be compressed
- Variable length

```
string_id = H5Tcopy(H5T_C_S1);
H5Tset_size(string_id, H5T_VARIABLE);
```

 - Overhead as for all VL datatypes (later)
 - Compression will not be applied to actual data

Bitfield Datatype

- C bitfield
- Bitfield – sequence of bytes packed in some integer type
- Examples of Predefined Datatypes
 - `H5T_NATIVE_B64` – native 8 byte bitfield
 - `H5T_STD_B32` – standard 2 bytes bitfield
- Created by copying predefined bitfield type and setting precision, offset and padding
- Use n-bit filter to store significant bits only

Bitfield Datatype

Example:

LE

0-padding



Offset 3

Precision 11

Opaque Datatype

- Datatype that cannot be described by any other HDF5 datatype
- Element treated as a blob of data and not interpreted by the library
- Identified by
 - Size
 - Tag (ASCII string)

Reference Datatype

- Reference to an HDF5 object
 - Pointers to Groups, datasets, and named datatypes in a file
 - Predefined datatype H5T_STD_REG_OBJ
 - H5Rcreate
 - H5Rdereference
- Reference to a dataset region (selection)
 - Pointer to the dataspace selection
 - Predefined datatype H5T_STD_REF_DSETREG
 - H5Rcreate
 - H5Rdereference

Enumeration Datatype

- Constructed after C/C++ enum type
- Name-value pairs
 - Name –ascii string
 - Value – of any HDF5 integer type
 - H5Tcreate
 - Creates the type based on integer type
 - H5Tinsert
 - Inserts name-value pairs
 - Order of insertion is not important
 - Two types are equal if they have the same pairs

Composite atomic HDF5 Datatypes

Array Datatype

- Element is multidimensional array of elements
- Base type can be of any HDF5 datatypes
- Example
 - Time series of speed ($v_1(t)$, $v_2(t)$, $v_3(t)$)
 - Speed vector (v_1 , v_2 , v_3) – all three components are needed; no subsetting by vector component (e.g. by v_1)

HDF5 Fixed and Variable length array storage



HDF5 Variable Length Datatypes

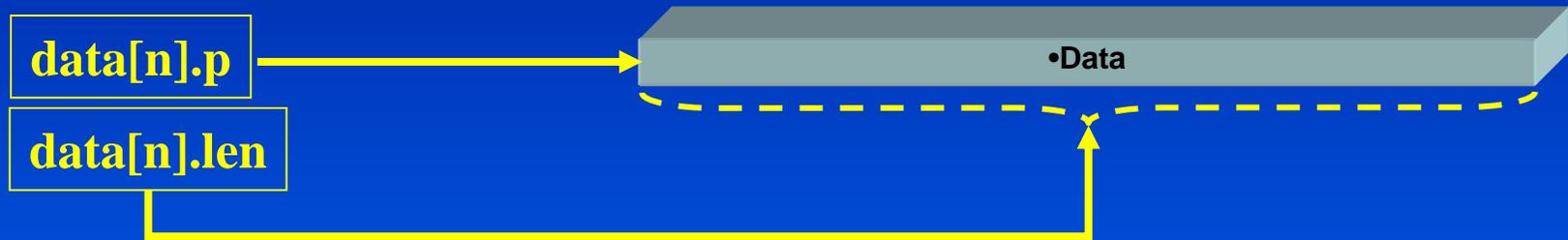
Programming issues

- Each element is represented by C struct

```
typedef struct {  
    size_t length;  
    void *p;  
} hvl_t;
```
- Base type can be any HDF5 type
- `H5Tvlen_create(base_type)`

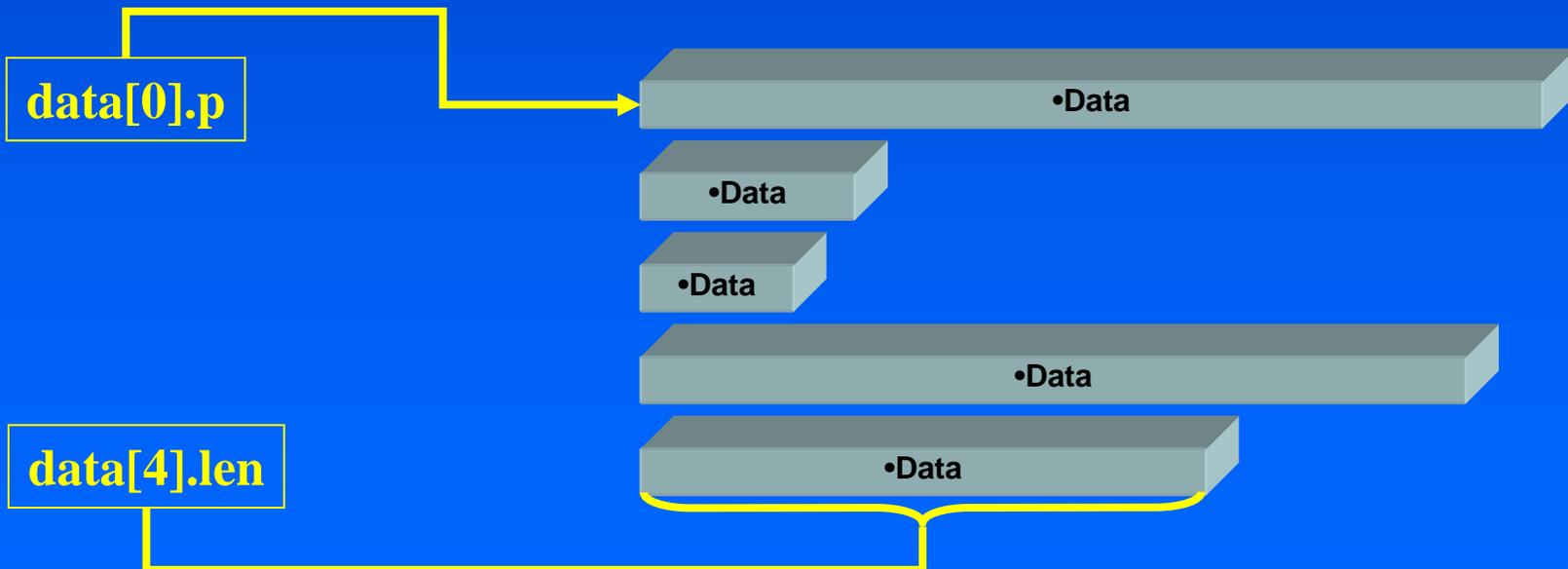
Creation of HDF5 Variable length array

hvl_t data[LENGTH]



Creation of HDF5 Variable length array

```
hvl_t data[LENGTH];  
  
for(i=0; i<LENGTH; i++) {  
data[i].p=HDmalloc((i+1)*sizeof(unsigned int));  
data[i].len=i+1;  
}  
  
tv1 = H5Tvlen_create (H5T_NATIVE_UINT);
```

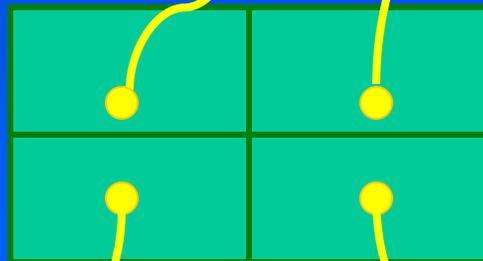


HDF5 Variable Length Datatypes Storage

Raw data



Global heap



Dataset with variable length datatype



Reading HDF5 Variable length array

When size and base datatype are known:

```
hvl_t  rdata[LENGTH];  
  
/* Discover the type in the file */  
tv1 = H5Tvlen_create (H5T_NATIVE_UINT);  
ret = H5Dread(dataset,tv1,H5S_ALL,H5S_ALL,  
              H5P_DEFAULT, rdata);  
  
/* Reclaim the read VL data */  
  
ret=H5Dvlen_reclaim(tv1,H5S_ALL,H5P_DEFAULT,rdata);
```

Reading HDF5 Variable length array

When size is unknown:

```
hvl_t  *rdata;

ret=H5Dvlen_get_buf_size(dataset,tvl,H5S_ALL,&size);

rdata = (hvl_t *)malloc(size);

ret = H5Dread(dataset,tvl,H5S_ALL,H5S_ALL,
              H5P_DEFAULT, rdata);

...

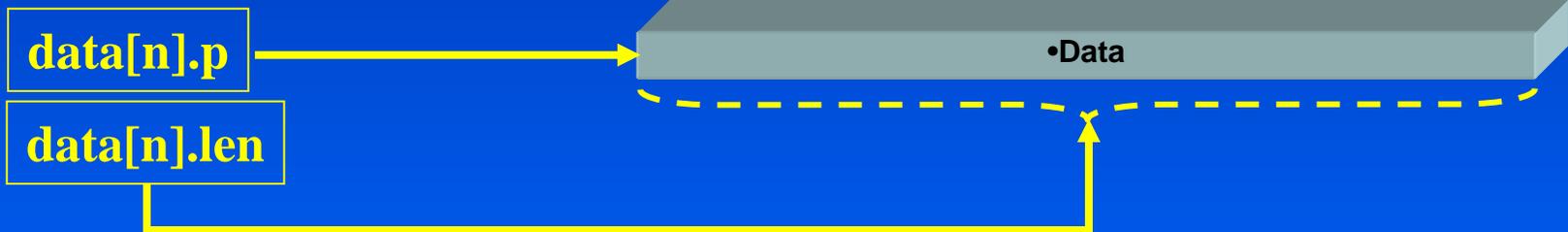
/* Reclaim the read VL data */
ret=H5Dvlen_reclaim(tvl,H5S_ALL,H5P_DEFAULT,rdata);

free(rdata);
```

Freeing HDF5 Variable length array

H5Dvlen_reclaim

free



Compound HDF5 Datatypes

HDF5 Compound Datatypes

- Compound types
 - Comparable to C structs
 - Members can be atomic or compound types
 - Members can be multidimensional
 - Can be written/read by a field or set of fields
 - *Non all data filters can be applied* (shuffling, SZIP)
 - H5Tcreate(H5T_COMPOUND), H5Tinsert calls to create a compound datatype
 - See H5Tget_member* functions for discovering properties of the HDF5 compound datatype

HDF5 Compound Datatypes

Creating and writing compound dataset

```
typedef struct s1_t {
    int a;
    float b;
    double c;
} s1_t;

s1_t    s1[LENGTH];

/* Initialize the data */
for (i = 0; i < LENGTH; i++) {
    s1[i].a = i;
    s1[i].b = i*i;
    s1[i].c = 1./(i+1);
}
```

HDF5 Compound Datatypes

Creating and writing compound dataset

```
/* Create datatype in memory. */  
  
s1_tid = H5Tcreate (H5T_COMPOUND, sizeof(s1_t));  
H5Tinsert(s1_tid, "a_name", HOFFSET(s1_t, a),  
          H5T_NATIVE_INT);  
H5Tinsert(s1_tid, "c_name", HOFFSET(s1_t, c),  
          H5T_NATIVE_DOUBLE);  
H5Tinsert(s1_tid, "b_name", HOFFSET(s1_t, b),  
          H5T_NATIVE_FLOAT);
```

Note:

- Use **HOFFSET** macro instead of calculating offset by hand
- Order of **H5Tinsert** calls is not important if **HOFFSET** is used

HDF5 Compound Datatypes

Creating and writing compound dataset

```
/* Create dataset and write data */  
  
dataset = H5Dcreate(file, DATASETNAME, s1_tid, space,  
                   H5P_DEFAULT);  
status  = H5Dwrite(dataset, s1_tid, H5S_ALL, H5S_ALL,  
                   H5P_DEFAULT, s1);
```

Note:

- In this example memory and file datatypes are the same
- Type is not packed
- Use `H5Tpack` to save space in the file

```
s2_tid = H5Tpack(s1_tid);  
status = H5Dcreate(file, DATASETNAME, s2_tid, space,  
                   H5P_DEFAULT);
```

File content with h5dump

```
HDF5 "SDScompound.h5" {
GROUP "/" {
  DATASET "ArrayOfStructures" {
    DATATYPE {
      H5T_STD_I32BE "a_name";
      H5T_IEEE_F32BE "b_name";
      H5T_IEEE_F64BE "c_name"; }
    DATASPACE { SIMPLE ( 10 ) / ( 10 ) }
    DATA {
      {
        [ 0 ],
        [ 0 ],
        [ 1 ]
      },
      {
        [ 1 ],
        [ 1 ],
        [ 0.5 ]
      },
      {
        [ 2 ],
        [ 4 ],
        [ 0.333333 ]
      },
      ...
    }
  }
}
```

HDF5 Compound Datatypes

Reading compound dataset

```
/* Create datatype in memory and read data. */  
  
dataset      = H5Dopen(file, DATSETNAME);  
s2_tid       = H5Dget_type(dataset);  
mem_tid      = H5Tget_native_type (s2_tid);  
status       = H5Dread(dataset, mem_tid, H5S_ALL,  
                        H5S_ALL, H5P_DEFAULT, s1);
```

Note:

We could construct memory type as we did in writing example

For general applications we need discover the type in the file to guess the structure to read to

HDF5 Compound Datatypes

Reading compound dataset: subsetting by fields

```
typedef struct ss_t {
    double a;
    float b;
} ss_t;
ss_t ss[LENGTH];
...
ss_tid = H5Tcreate (H5T_COMPOUND, sizeof(ss_t));
H5Tinsert(s1_tid, "c_name", HOFFSET(ss_t, c),
          H5T_NATIVE_DOUBLE);
H5Tinsert(s1_tid, "b_name", HOFFSET(ss_t, b),
          H5T_NATIVE_FLOAT);
...
status = H5Dread(dataset, ss_tid, H5S_ALL,
                 H5S_ALL, H5P_DEFAULT, ss);
```

Discovering HDF5 Datatypes

Discovering Datatype

1. get class
2. get size
3. if numeric atomic
 - A. *get precision, sign, padding, mantissa, exponent, etc*
 - B. *allocate space and read data*
4. if VL , enum or array
 - A. *get super class; go to 2*
5. if compound
 - A. *get number of members and members' offsets*
 - B. *go to 1*

HDF5 Compound Datatypes

Discovering Datatype

```
s1_tid = H5Dget_type(dataset);
if (H5Tget_class(s1_tid) == H5T_COMPOUND)
{
    sz = H5Tget_size(s1_tid);
    nmemb = H5Tget_nmembers(s1_tid);
    for (i =0; i < nmemb; i++) {
        s2_tid = H5Tget_member_type(s1_tid,i);
        H5Tget_member_name(s1_tid,i);
        H5Tget_member_offset(s1_tid,i),
        if (H5Tget_class(s2_tid) == H5T_COMPOUND) {
            /* recursively analyze the nested type. */
        }
        else if (H5Tget_class(s2_tid) == H5T_ARRAY) {
            sz2 = H5Tget_size(s2_tid);
            H5Tget_array_dims(s2_tid,dim,NULL);
            s3_tid = H5Tget_super(s2_tid);
        }
    }
}
```

HDF Information

- HDF Information Center
 - <http://hdf.ncsa.uiuc.edu/>
- HDF Help email address
 - hdfhelp@ncsa.uiuc.edu
- HDF users mailing list
 - hdfnews@ncsa.uiuc.edu