



# HDF5 Advanced Topics

## Outline

- Partial I/O and dataset selections
- Chunking and filters
- Datatypes
  - Overview
  - Variable length datatype
  - Compound datatype
  - Object and dataset region references



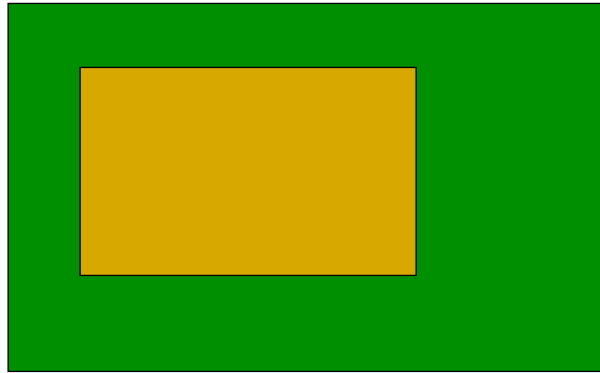
# Partial I/O and Selections

## What is a selection?

- Selection describes elements of a dataset that participate in partial I/O
  - Hyperslab selection
  - Point selection
  - Results of Set Operations on hyperslab selections or point selections (union, difference, ...)
- Used by sequential and parallel HDF5



## Example of a hyperslab selection



September 9, 2008

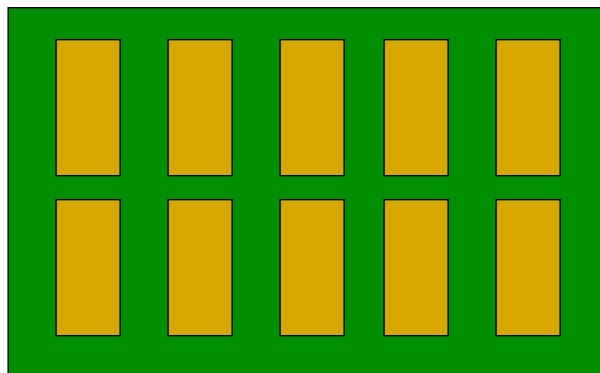
SPEEDUP Workshop - HDF5 Tutorial

5

[www.hdfgroup.org](http://www.hdfgroup.org)



## “Regular” hyperslab selection



September 9, 2008

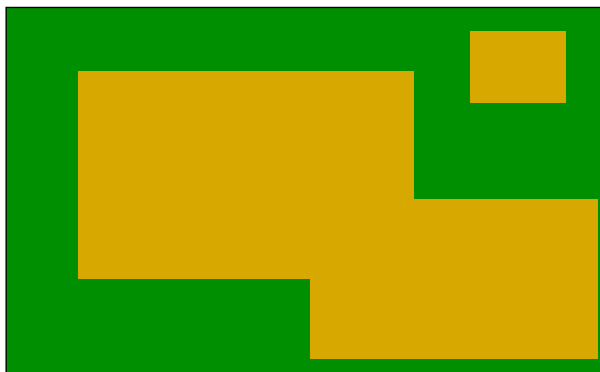
SPEEDUP Workshop - HDF5 Tutorial

6

[www.hdfgroup.org](http://www.hdfgroup.org)



## “Irregular” hyperslab selection



September 9, 2008

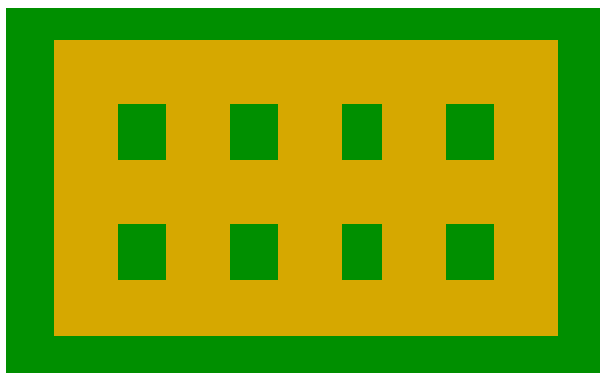
SPEEDUP Workshop - HDF5 Tutorial

7

[www.hdfgroup.org](http://www.hdfgroup.org)



## More selections



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

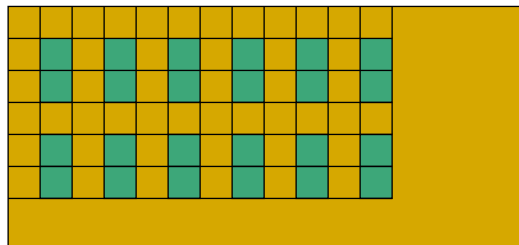
8

[www.hdfgroup.org](http://www.hdfgroup.org)



## Hyperslab description

- Offset - starting location of a hyperslab (1,1)
- Stride - number of elements that separate each block (3,2)
- Count - number of blocks (2,6)
- Block - block size (2,1)
- *Everything is “measured” in number of elements*



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

9

www.hdfgroup.org



## H5Sselect\_hyperslab

***space\_id*** Identifier of dataspace  
***op*** Selection operator  
H5S\_SELECT\_SET or H5S\_SELECT\_OR  
***offset*** Array with starting coordinates of hyperslab  
***stride*** Array specifying which positions along a dimension to select  
***count*** Array specifying how many blocks to select from the dataspace, in each dimension  
***block*** Array specifying size of element block  
(NULL indicates a block size of a single element in a dimension)

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

10

www.hdfgroup.org



## Reading/Writing Selections

1. Open the file
2. Open the dataset
3. Get file dataspace
4. Create a memory dataspace (data buffer)
5. Make the selection(s)
6. Read from or write to the dataset
7. Close the dataset, file dataspace, memory dataspace, and file



## Example : reading two rows

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

Data in file  
4x6 matrix

Buffer in memory  
1-dim array of length 14

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
----	----	----	----	----	----	----	----	----	----	----	----	----	----



## Example: reading two rows

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

```
offset = {1,0}  
count  = {2,6}  
block  = {1,1}  
stride = {1,1}
```

```
file_space = H5Dget_space (dataset);  
H5Sselect_hyperslab (file_space, H5S_SELECT_SET,  
                    offset, NULL, count, NULL)
```



## Example: reading two rows

```
offset = {1}  
count  = {12}
```

-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
----	----	----	----	----	----	----	----	----	----	----	----	----	----

```
memspace = H5Screate_simple(1, 14, NULL);  
H5Sselect_hyperslab (memspace, H5S_SELECT_SET,  
                    offset, NULL, count, NULL)
```



## Example: reading two rows

1	2	3	4	5	6
7	8	9	10	11	12
13	14	15	16	17	18
19	20	21	22	23	24

```
H5Dread (... , ..., memspace, filespace, ..., ...);
```

-1	7	8	9	10	11	12	13	14	15	16	17	18	-1
----	---	---	---	----	----	----	----	----	----	----	----	----	----

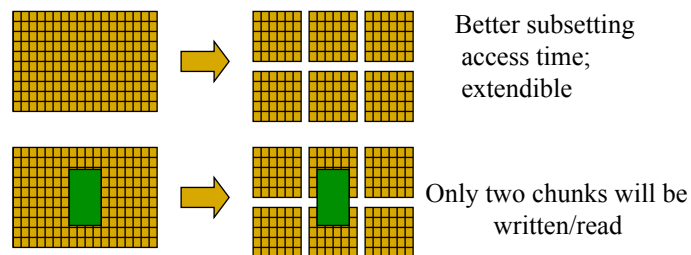


## Chunking in HDF5



## HDF5 chunking

- Chunked storage layout is needed for
  - ✓ Extendible datasets
  - ✓ Enabling compression and other filters
  - ✓ Improving partial I/O for big datasets



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

17

[www.hdfgroup.org](http://www.hdfgroup.org)



## Creating chunked compressed dataset

1. Create a dataset creation property list
2. Set property list to use chunked storage layout
3. Set property list to use filters
4. Create dataset with the above property list

```
plist = H5Pcreate(H5P_DATASET_CREATE);  
        H5Pset_chunk(plist, rank, ch_dims);  
        H5Pset_deflate(plist, 9);  
  
dset_id = H5Dcreate (... , "Chunked", ..., plist);  
        H5Pclose(plist);
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

18

[www.hdfgroup.org](http://www.hdfgroup.org)



## HDF5 filters

- HDF5 filters modify data during I/O operations
- Available filters:
  1. Checksum (H5Pset\_fletcher32)
  2. Shuffling filter (H5Pset\_shuffle)
  3. Data transformation (in 1.8.0)
  4. Compression
    - Scale + offset (in 1.8.0)
    - N-bit (in 1.8.0)
    - GZIP (deflate), SZIP (H5Pset\_deflate, H5Pset\_szip)
    - User-defined filters (BZIP2)
      - Example of a user-defined compression filter can be found  
<http://www.hdfgroup.uiuc.edu/papers/papers/bzip2/>



## Writing or reading to/from chunked dataset

1. Use the same set of operation as for contiguous dataset
2. Selections do not need to coincide precisely with the chunks
3. Chunking mechanism is transparent to application
4. *Chunking and compression parameters can affect performance*

```
H5Dopen (...);  
.....  
H5Sselect_hyperslab (...);  
.....  
H5Dread (...);
```



## h5zip.c example

Creates a compressed 1000x20 integer dataset in a file

```
%h5dump -p -H zip.h5
HDF5 "zip.h5" {
  GROUP "/" {
    GROUP "Data" {
      DATASET "Compressed_Data" {
        DATATYPE  H5T_STD_I32BE
        DATASPACE  SIMPLE { ( 1000, 20 ).....
        STORAGE_LAYOUT {
          CHUNKED ( 20, 20 )
          SIZE 5316
        }
      }
    }
  }
}
```



## h5zip.c example

```
FILTERS {
  COMPRESSION DEFLATE { LEVEL 6 }
}
FILLVALUE {
  FILL_TIME H5D_FILL_TIME_IFSET
  VALUE 0
}
ALLOCATION_TIME {
  H5D_ALLOC_TIME_INCR
}
}
}
```



## h5zip.c example (bigger chunk)

Creates a compressed integer dataset 1000x20 in the zip.h5 file

```
h5dump -p -H zip.h5
```

```
HDF5 "zip.h5" {  
  GROUP "/" {  
    GROUP "Data" {  
      DATASET "Compressed_Data" {  
        DATATYPE H5T_STD_I32BE  
        DATASPACE SIMPLE { ( 1000, 20 ).....  
        STORAGE_LAYOUT {  
          CHUNKED ( 200, 20 )  
          SIZE 2936  
        }  
      }  
    }  
  }  
}
```



## Chunking basics to remember

- Chunking creates storage overhead in the file
- Performance is affected by
  - Chunking and compression parameters
  - Chunking cache size (`H5Pset_cache` call)
- Some hints for getting better performance
  - Use chunk size no smaller than block size (4k) on your system
  - Use compression method appropriate for your data
  - Avoid using selections that do not coincide with the chunking boundaries

# Selections and I/O Performance

- Next slides show the performance effects of using different access patterns and storage layouts.
- We use three test cases which consist of writing a selection to a rectangular array where the datatype of each element is a char.
- Data is stored in row-major order.
- Tests were executed on THG *Linux x86\_64* box using *h5perf\_serial* and HDF5 version 1.8.



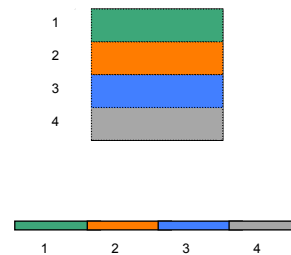
## Serial benchmarking tool

- A new benchmarking tool, *h5perf\_serial*, is under development.
- Some features implemented at this time are:
  - ✓ Support for POSIX and HDF5 I/O calls.
  - ✓ Support for datasets and buffers with multiple dimensions.
  - ✓ Entire dataset access using a single or several I/O operations.
  - ✓ Selection of contiguous and chunked storage for HDF5 operations.



## Contiguous storage (Case 1)

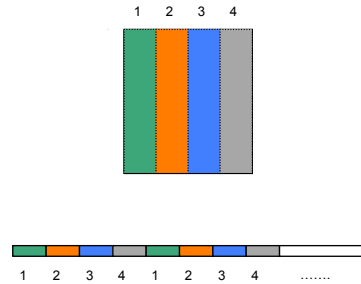
- Rectangular dataset of size 48K x 48K, with write selections of 512 x 48K.
- HDF5 storage layout is contiguous.
- Good I/O pattern for POSIX and HDF5 because each selection is contiguous.
- POSIX: 5.19 MB/s
- HDF5: 5.36 MB/s





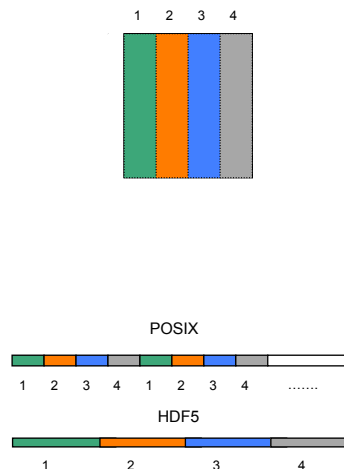
## Contiguous storage (Case 2)

- Rectangular dataset of 48K x 48K, with write selections of 48K x 512.
- HDF5 storage layout is contiguous.
- Bad I/O pattern for POSIX and HDF5 because each selection is noncontiguous.
- POSIX: 1.24 MB/s
- HDF5: 0.05 MB/s



## Chunked storage

- Rectangular dataset of 48K x 48K, with write selections of 48K x 512.
- HDF5 storage layout is chunked. Chunks and selections sizes are equal.
- Bad I/O case for POSIX because selections are noncontiguous.
- Good I/O case for HDF5 since selections are contiguous due to chunking layout settings.
- POSIX: 1.51 MB/s
- HDF5: 5.58 MB/s





## Conclusions

- Access patterns with small I/O operations incur high latency and overhead costs many times.
- Chunked storage may improve I/O performance by affecting the contiguity of the data selection.



## HDF5 datatypes



## HDF5 Datatypes

- Can we easily write data to the same file from different languages and on different platforms?
- Can we easily read data back anywhere?
- Can we interpret the data we read back?
- HDF5's answer: YES

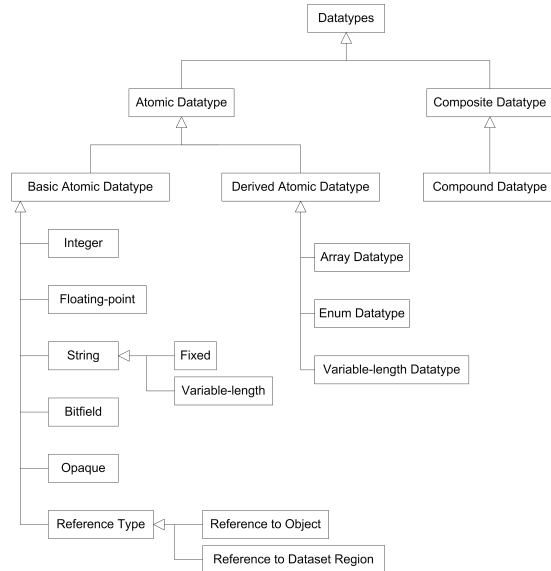


## HDF5 Datatypes

- HDF5 supports a rich set of pre-defined datatypes as well as the creation of an unlimited variety of complex user-defined datatypes.
- Datatype description is stored in the HDF5 file *with* the data
- Datatype definitions include information such as *byte order (endianess), size, and floating point representation*, to fully describe how the data is stored, insuring portability to other platforms.
- Datatype definitions can be *shared* among objects in an HDF file, providing a powerful and efficient mechanism for describing data.



## Hierarchy of the HDF5 datatype classes



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

35

www.hdfgroup.org



## Operations on HDF5 Datatypes

- Create
  - Derived and compound datatypes only
- Copy
  - All datatypes
- Commit (save in a file to share between different datasets)
  - All datatypes
- Open
  - Committed datatypes only
- Discover properties (size, number of members, base type)
- Close

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

36

www.hdfgroup.org



# Basic atomic HDF5 datatypes



## Basic Atomic Datatypes

- Atomic types classes
  - integers & floats
  - strings (fixed and variable size)
  - pointers - references to objects/dataset regions
  - 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 filed for integers



## Example: Integers and Floats

- Integer - But which one?
  - How many bytes: 1, 2, 4, or 8?
  - Little or big-endian?
  - Signed or unsigned?
- Real or Float - But which one?
  - IEEE or VAX G-floating?
  - Little or big-endian?
  - 32 or 64-bit ?
- HDF5 takes care of storing data in a prescribed way and getting it back to the user in a desired way.



## Advantages of HDF5

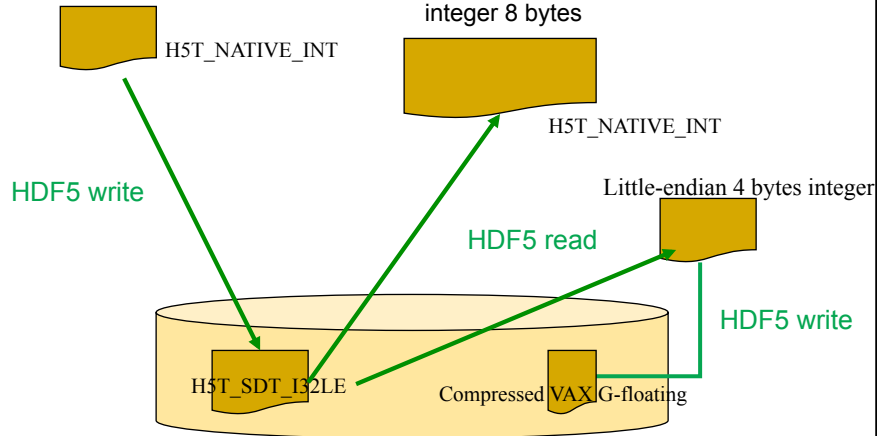
- HDF5 knows the format in which data is written
- HDF5 gives back data in the requested format
- HDF5 is aware of “native-ness” (endianess, sizes, architectures, etc.)



## Example: data in HDF5

Array of integers on Linux platform  
Native integer is little-endian, 4 bytes

Array of integers on Solaris platform  
Native integer is big-endian,  
Fortran compiler uses -i8 flag to make  
integer 8 bytes



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

41

www.hdfgroup.org



## HDF5 Predefined Datatypes

- HDF5 Library provides predefined datatypes (symbols) for describing all basic atomic classes except opaque
  - H5T\_<arch>\_<base>
  - Examples:
    - H5T\_IEEE\_F64LE
    - H5T\_STD\_I32BE
    - H5T\_C\_S1
    - H5T\_STD\_B32LE
    - H5T\_STD\_REF\_OBJ, H5T\_STD\_REF\_DSETREG
    - H5T\_NATIVE\_INT
- *Predefined datatypes do not have constant values; initialized when library is initialized*

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

42

www.hdfgroup.org



## When to use HDF5 Predefined Datatypes?

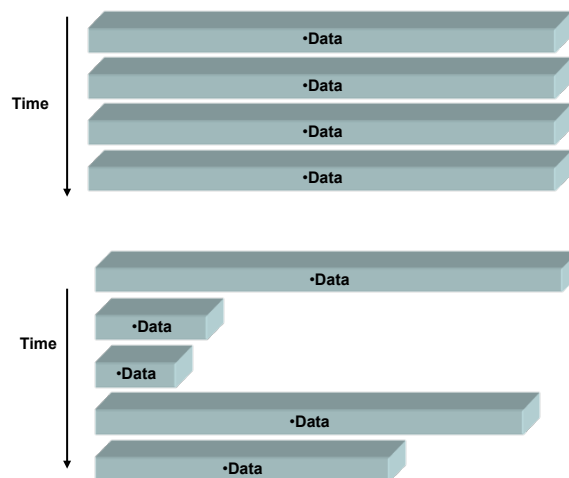
- In datasets and attributes creation operations
  - Argument to H5Dcreate or to H5Acreate

```
H5Dcreate(file_id, "/dset", H5T_STD_I32BE, dataspace_id, H5P_DEFAULT);
```

```
H5Dcreate(file_id, "/dset", H5T_NATIVE_INT, dataspace_id, H5P_DEFAULT);
```
- In datasets and attributes I/O operations
  - Argument to H5Dwrite/read, H5Awrite/read
  - **Always use `H5T_NATIVE_*` types to describe data in memory**
- 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*



## HDF5 Fixed and Variable length array storage





## Storing strings in HDF5 (string.c)

- 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
- See string.c for how to allocate buffer for reading data for fixed and variable length strings



## HDF5 variable length datatypes

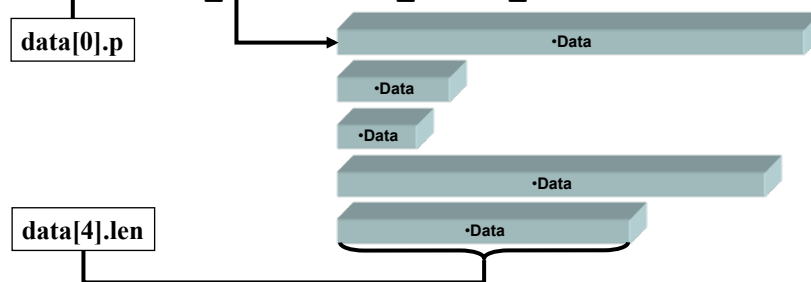
- Each element is represented by C struct

```
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];  
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);
```



September 9, 2008

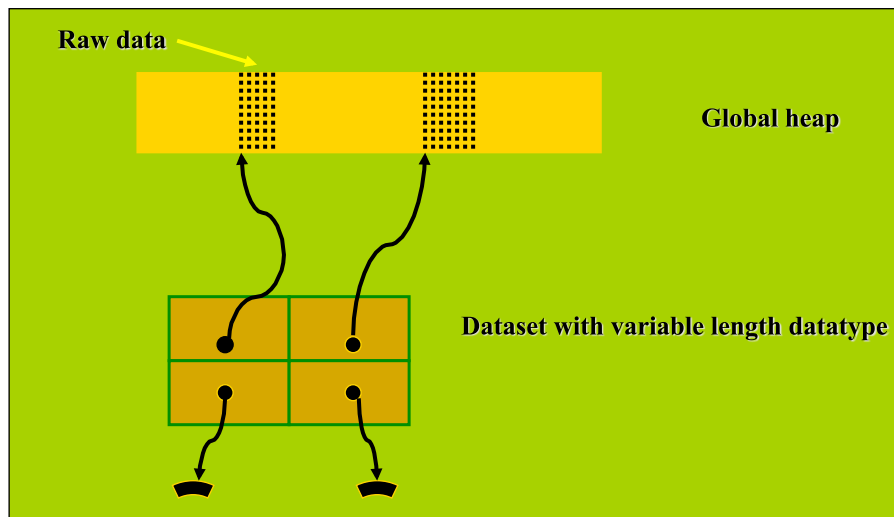
SPEEDUP Workshop - HDF5 Tutorial

47

www.hdfgroup.org



## HDF5 Variable Length Datatypes Storage



September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

48

www.hdfgroup.org



## 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);
```



## 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\_B32LE – standard 4 bytes bitfield
- Created by copying predefined bitfield type and setting precision, offset and padding
- Use n-bit filter to store significant bits only





## Example

a_name (integer)	b_name (float)	c_name (double)
0	0.	1.0000
1	1.	0.5000
2	4.	0.3333
3	9.	0.2500
4	16.	0.2000
5	25.	0.1667
6	36.	0.1429
7	49.	0.1250
8	64.	0.1111
9	81.	0.1000

### Multiple ways to store a table

Dataset for each field

Dataset with compound datatype

If all fields have the same type:

2-dim array

1-dim array of array datatype

continued.....

### Choose to achieve your goal!

*How much overhead each type of storage will create?*

*Do I always read all fields?*

*Do I need to read some fields more often?*

*Do I want to use compression?*

*Do I want to access some records?*



## 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)*



## HDF5 Compound Datatypes

- Which APIs to use?
  - H5TB APIs
    - Create, read, get info and merge tables
    - Add, delete, and append records
    - Insert and delete fields
    - Limited control over table's properties (i.e. only GZIP compression, level 6, default allocation time for table, extendible, etc.)
  - PyTables <http://www.pytables.org>
    - Based on H5TB
    - Python interface
    - Indexing capabilities
  - HDF5 APIs
    - H5Tcreate(H5T\_COMPOUND), H5Tinsert calls to create a compound datatype
    - H5Dcreate, etc.
    - See H5Tget\_member\* functions for discovering properties of the HDF5 compound datatype

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

55

[www.hdfgroup.org](http://www.hdfgroup.org)



## Creating and writing compound dataset

### h5\_compound.c example

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

s1_t s1[LENGTH];
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

56

[www.hdfgroup.org](http://www.hdfgroup.org)



## 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

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

57

www.hdfgroup.org



## 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);
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

58

www.hdfgroup.org



## 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 ]
        },
        ...
      }
    }
  }
}
```



## Reading compound dataset

```
/* Create datatype in memory and read data. */

dataset      = H5Dopen(file, DATASETNAME);
s2_tid       = H5Dget_type(dataset);
mem_tid      = H5Tget_native_type(s2_tid);
s1 = malloc(sizeof(mem_tid)*number_of_elements)
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 to discover the type in the file, find out corresponding memory type, allocate space and do read



## Reading compound dataset: subsetting by fields

```
typedef struct s2_t {
    double c;
    int    a;
} s2_t;
s2_t s2[LENGTH];
...
s2_tid = H5Tcreate (H5T_COMPOUND, sizeof(s2_t));
H5Tinsert(s2_tid, "c_name", HOFFSET(s2_t, c),
          H5T_NATIVE_DOUBLE);
H5Tinsert(s2_tid, "a_name", HOFFSET(s2_t, a),
          H5T_NATIVE_INT);
...
status = H5Dread(dataset, s2_tid, H5S_ALL,
                 H5S_ALL, H5P_DEFAULT, s2);
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

61

[www.hdfgroup.org](http://www.hdfgroup.org)



## Reading compound dataset: subsetting by fields

### Another way to create a compound datatype

```
#include H5LTpublic.h
...
s2_tid = H5LTtext_to_dtype(
    "H5T_COMPOUND
    {H5T_NATIVE_DOUBLE \"c_name\";
    H5T_NATIVE_INT \"a_name\";
    }",
    H5LT_DDL);
```

September 9, 2008

SPEEDUP Workshop - HDF5 Tutorial

62

[www.hdfgroup.org](http://www.hdfgroup.org)



# Questions?