RFC: Changing Tools to Support Sparse Data in HDF5

John Mainzer (john.mainzer@lifeboat.llc),
Elena Pourmal (elena.pourmal@lifeboat.llc)
Lifeboat, LLC

Allen Byrne (byrn@hdfgroup.org)
Glenn Song (gsong@hdfgroup.org)
The HDF Group

In general, all command-line tools should work with sparse data as they work with the dense data. Of course, this statement requires confirmation. Existing tools tests and files have to be updated to include sparse datasets.

In this RFC we outline the changes to a few tools that would require changes to the user's interface, i.e., additional flags to display locations of the defined elements or to the output to specify the usage of sparse chunk storage. Here we only outline the major changes and will leave the exact specifications to another tools specific RFC.

1 Tools that require updating

We propose a plan for updating HDF5 command-line tools to accommodate structured chunk data in the following order. The first pass at updating the tools should include a limited subsection of the tools to focus on. Specifically, h5dump and h5ls are good targets for this. The changes required for h5dump are extensive, but due to the useful nature of h5dump as a tool, it should be prioritized. Once the changes are made for h5dump, the changes to get h5ls working will be much simpler. This RFC proposes a detailed plan for updating each tool and also covers the tools that do not need to be changed.

1.1 h5dump

1.1.1 DDL Changes

The DDL used for h5dump output should be updated to use the new “structured chunk” type. It will display the dimensions of the dataset, then will show the specific type of structured chunk used, then it will display any sections. Specifically, it could be sparse chunk or variable length chunk. Within each section, it will display the section number, the section name, the size, and then the compression ratio. The section name describes the section and is either sparse selection, sparse fixed data, or sparse variable length.

```
<structured_chunk_layout> ::= STRUCTURED_CHUNK <dims> {
                    <structured_chunk_type>
                    <sections>
                } opt

<dims> ::= <size> | <size>, <dims>

<structured_chunk_type> ::= SPARSE_CHUNK | VL_CHUNK

<sections> ::= SECTION <section_number> <section_name> <filter_ratio> {
                    <compression_filters>
                } opt

<section_number> ::= <int_value>

<section_name> ::= SPARSE_SELECTION | SPARSE_FIXED_DATA | SPARSE_VL
```

Additionally, we must update the data section of the DDL to accommodate the changes to how data is displayed.

```
<sparse_data> ::= <sparse_defined>
                            <data>opt

<sparse_defined> ::= DEFINED_SPARSE_DATA <data_region_data_type>
```

1.1.2 Output example using the updated DDL

We provide here an example of what the output from h5dump might look like for a dataset created in Example 1 that uses sparse storage with compression. The "SPARSE_CHUNK" keyword indicates the new storage type. The rest of the output is similar to the output for a dataset with chunked storage and GZIP compression applied to each section of the structured chunk.

```
$ hdf5/bin/h5dump -properties -header h5sparse.h5
HDF5 "h5sparse.h5" {
GROUP "/" {
   DATASET "DS1" {
      DATATYPE  H5T_STD_I32LE
      DATASPACE  SIMPLE { ( 32, 64 ) / ( 32, 64 ) }
      STORAGE_LAYOUT {
         STRUCTURED_CHUNK ( 5, 9 ) {
             SECTION 0 <section_name>[1] {
                 SIZE 100 (2:1 COMPRESSION)
                 FILTERS {
                     COMPRESSION DEFLATE { LEVEL 9 }
                }
             }
             SECTION 1 <section_name> {
                SIZE 5018 (1.633:1 COMPRESSION)
                FILTERS {
                    COMPRESSION SZIP {
                        PIXELS_PER_BLOCK 4
                        MODE K13
                        CODING NEAREST NEIGHBOUR
                        BYTE_ORDER LSB
                        HEADER RAW
                                    }
                }
             )
         }
      }

      FILLVALUE {
         FILL_TIME H5D_FILL_TIME_IFSET
         VALUE  0
      }
      ALLOCATION_TIME {
         H5D_ALLOC_TIME_INCR
      }
   }
}
}
```

[1] We may provide sections names that are symbols used to set compression flag, e.g., H5Z_FLAG_SPARSE_SELECTION

1.1.3 New flags and options

We should note here that existing sub-setting flags can be used to specify the bounding box to print sparse data as usual using fill values for data that is not defined. A new flag, `--sparse-locations`, will be introduced to print the locations of the defined elements within the dataset. The output will contain the coordinates of the individual elements or the coordinates of the "upper left" and "lower right" simple hyperslab if all points are defined in that hyperslab. Such formats are already used to print hyperslab and point selections (see the format used to print `REGION_TYPE BLOCK`, e.g. `(0,0) – (8,8)` will represent a subarray of the size 9x9 located in the left upper corner of a matrix, and the format used to print `REGION_TYPE POINT`[2], e.g., `(0,0),(1,1), …,(N,N)` will represent a diagonal elements of `N x N` matrix.

Figure 1 was stored as a dataset "Sparse" using sparse chunk layout, then the output for its data locations may look as shown here:

```
DATASET "Sparse" {
            DATATYPE  H5T_STD_U8BE
            DATASPACE  SIMPLE { ( 13, 10 ) / ( 13, 10 ) }
            DATA {
                DEFINED_SPARSE_DATA BLOCK (2,2)-(4,7)
                 DEFINED_SPARSE_DATA BLOCK (6,0)-(6,2)
                 DEFINED_SPARSE_DATA POINT (5,9), (11, 1), (12,8)
            }
}
```

Figure 1: "Sparse" dataset shown by h5dump output above

Row/column index	0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0	0	0	
2	0	0	66	69	72	75	78	81	0	0
3	0	0	96	99	102	105	108	111	0	0
4	0	0	126	129	132	135	138	141	0	0
5	0	0	0	0	0	0	0	0	0	2
6	100	0	-100	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	0	0	0	0	0	0
10	0	0	0	0	0	0	0	0	0	0
11	0	1	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	3	0

[2] Initial implementation will work with the hyperslab selections only. We use this as an example if in the future implementation will be extended to allow the point selections. Currently one will need to use hyperslab selection to specify one element, i.e., the line for the individual points `REGION_TYPE POINT (5,9), (11, 1), (12,8)` in h5dump output above will become

REGION_TYPE BLOCK (5,9) - (5,9)
REGION_TYPE BLOCK (11,1) - (11,1)
REGION_TYPE BLOCK (12,8) - (12,8)

There settings may also be added to the `--sparse-locations` option to further specify what kind of data will be output.

Another flag, `--sparse-data`, will be introduced to add functionality for printing the location of the data together with the data itself. The example output for `--sparse-data` is displayed below using the data from Figure 3. It splits the output between data blocks and data points. Displaying individual data points is currently out of scope for this project, and planned functionality only includes printing data blocks.

```
DATASET “Sparse" {
         DATATYPE  H5T_STD_U8BE
         DATASPACE  SIMPLE { ( 13, 10 ) / ( 13, 10 ) }
         DATA {
             DEFINED_SPARSE_DATA BLOCK  (2,2)-(4,7)
             DATA {
             (2,2)   66, 69, 72, 75, 78, 81,
             (3,2)   96, 99, 102, 105, 108, 111,
             (4,2)   126, 129, 132, 135, 138, 141
             }
             DEFINED_SPARSE_DATA BLOCK (6,0)-(6,2)
             DATA {
             (6,0)   100, 0, -100
             }
             DEFINED_SPARSE_DATA POINT (5,9), (11, 1), (12,8)
             DATA {
             (11,1) 1
             (5,9) 2
             (12,8) 3
             }
          }
      }
```

We will print the coordinates of the first element on each line of the output. In the example above, two `BLOCK` segments represent simple hyperslabs and their data, and the `POINT` segment shows the list of all point selections (e.g., coordinates of the elements and the values). An argument may be further added to print only a subset of the dataset.

1.1.4 Required code changes

In order to support the structured chunk storage, here is a proposed plan for changes that must be made.

In h5tools.h, there are new defines that must be created (insert at 1.14.4 L108), which will be used by h5dump for display purposes. These include the new layout type, the sections it can contain, and the current two types that it supports, sparse and variable length.

```
#define STRUCTURED          "STRUCTURED CHUNK"
#define SECTION             "SECTION"
```

```
#define SPARSE_CHUNK        "SPARSE+CHUNK"
#define VL_CHUNK            "VL_CHUNK"
```

In h5tools_dump_header_t, existing variables can be reused, but we must also add (insert at 1.14.4 L187-188):

```
const char *structchunkbegin;
const char *structchunkend;
```

In h5tool_format_t, new fields that are associated with structured chunks must be added to the field filters (insert at 1.14.4 L507) with a description of what the filters option is used for:

- 1: Indicates to print list of filters
- 0: Skip

In h5tools_dump.c,

- h5tools_dataformat structure will implement the changes identified above for h5tool_format_t (1.14.4 L93).
- h5tools_standardformat will implement the changes identified above for h5tools_dump_header_t (1.14.4 L166-167).
- h5tools_dump_dcpl(FILE *stream, const h5tool_format_t *info, h5tools_context_t *ctx, hid_t dcpl_id, hid_t type_id, hid_t dset_id) will require a new structured chunk case to be added to the switch case (1.14.4 L3181).

Please note that the FILTERS section of code should be moved to a new function, so that the filters OUTPUT can be reused by the structured chunk code. The steps for doing so are listed below:

The STORAGE_LAYOUT switch statement cases for structured chunk must be implemented. To do so, we must get the chunk size and other creation properties including filtering in int H5Pget_struct_chunk (hid_t plist_id, int max_dims, const hsize dim[], unsigned *flag).

Next, we must select elements in the dataset in the file by using:

- For chunk_idx = 0 to max_dims
  - herr_t H5Dget_struct_chunk_info (hid_t dset_id, hid_t fspace_id, hsize_t chunk_idx, const hsize_t *offset, H5D_struct_chunk_info_t *chunk_info, haddr_t *addr, hsize_t *chunk_size)
  - herr_t H5Dread_struct_chunk (hid_t dset_id, hid_t dxpl_id, const hsize_t *offset, H5D_struct_chunk_info_t *chunk_info, void *buf[])
    - If show_filters is true, call new function to display filter information.

For dump_dataset(hid_t did, const char *name, struct subset_t *sset) and h5tools_dump_data(FILE *stream, const h5tool_format_t *info, h5tools_context_t

*ctx, hid_t obj_id, int obj_data), current hyperslab selection API (i.e., no changes for read are required) must be updated. We must also get the selection of "defined" elements in the provided bounding box for hid_t H5Dget_defined (hid_t dataset_id, hid_t file_space_id, hid_t xfer_plist_id).

In h5dump, we will have to update h5dump.c. Specifically, we must update the doxygen to include the two new options for the tool.

- We need to add \li --sparse-locations Print the locations of the defined elements within the dataset. (h5dump.h 1.14.4 L67)
- We need to add \li --sparse-data Print the locations of the defined elements within the dataset and the associated data. (h5dump.h 1.14.4 L68)
- We should also add an example to the "Usage Examples" section at the end for how to print structured chunk data and how to use the new options added. (h5dump.h 1.14.4 L193)
- dump_opt_t structure will need new options for display settings for the new command-line arguments (h5dump.h 1.14.4 L81).

h5dump.c must also be updated to reflect the two new options for --sparse-locations and --sparse-data. A character must be chosen for both options (currently 'L' and 'L' are selected as examples).

h5_long_options l_opts[] must add {"sparse-locations", no_arg, 'L'} and {"sparse-data", no_arg, 'I'} (h5dump.c 1.14.4 L150). These new options should also be added to usage(const char *prog) (h5dump.c 1.14.4 L240). static struct handler_t * parse_command_line(int argc, const char *const *argv) (h5dump.c 1.14.4 L880) must also be updated to include the switch cases for the two command-line options.

XML support has languished and probably should be removed. At a minimum, just the storage layout info should be added to h5dump_xml.c.

1.1.5 Testing

Additionally, a test should be added to ensure that the output of h5dump is correct in the case of both sparse chunk and variable length data.

1.2 h5ls

1.2.1 Output example

Once implemented, the only required change will be to indicate the new type of storage "Structured Chunks" (vs. current "Chunks") as shown below:

```
% h5ls -rv tfilters.h5
Opened "tfilters.h5" with sec2 driver.
```

```
/                        Group
    Location:  1:96
    Links:     1
/all                     Dataset {20/20, 10/10}
    Location:  1:29336
    Links:     1
    Structured Chunks: {10, 5} 200 bytes
    Section 1 <section_name>
    Storage:   800 logical bytes, 458 allocated bytes, 174.67% utilization
    Filter-0:  shuffle-2 OPT {4}
    Filter-1:  szip-4 OPT {141, 4, 32, 5}
    Filter-2:  deflate-1 OPT {5}
    Filter-3:  fletcher32-3  {}
    Filter-4:  nbit-5 OPT {8, 1, 50, 1, 4, 0, 32, 0}
    Type:      native int
    Section 2 <section name>
    ……
```

1.2.2 New flags and options

Since `h5ls` doesn't have flags for specifying sub-setting and de-referencing region references and doesn't use corresponding output format as `h5dump`, we suggest no changes to the tool (i.e., any new flags) to print defined sparse data elements.

1.2.3 Required code changes

However, changes to the existing behavior are necessary for the addition of structured chunk. For `h5ls.h`, the help text for `\li <strong>--address</strong>` (h5ls.h 1.14.4 L97) will need to be added.

In `h5ls.c`, the updates to the help text must be made here as well. `static void dump_dataset_values(hid_t dset)` and `static herr_t dataset_list2(hid_t dset, const char H5_ATTR_UNUSED *name)` (h5ls.c 1.14.4 L1952) must both be updated to add a case for structured chunk by including the information that must be displayed.

1.2.4 Testing

Tests should be added to make sure that the tool's output has been correctly updated to account for the new structured chunk storage layout.

1.3 h5stat

1.3.1 Output example

We will need to update the tool's output to display the number of the datasets with structured chunk layout. For example, if a file contains three sparse datasets along with the datasets with other storage layouts, the output will look like this:

```
Dataset layout information:
     Dataset layout counts[COMPACT]: …
     Dataset layout counts[CONTIG]: …
     Dataset layout counts[CHUNKED]: …
     Dataset layout counts[STRUCTURED CHUNK SPARSE]: 3
     Dataset layout counts[VIRTUAL]: …
     …..
```

For structured chunk dataset filter information will include all filters applied to the different sections. If the same filter is applied to multiple sections of the structured chunk, it is counted only once for the dataset.

1.3.2 New flags and options

No new flags or options are required for h5stat, since the only thing that changes should be the displayed text.

1.3.3 Required code changes

In regards to code changes that must be made:

In h5stat.c,

- `static herr_t dataset_stats(iter_t *iter, const char *name, const H5O_info2_t *oi, const H5O_native_info_t *native_oi)`
  - Uses `H5D_layout_t` and lists layout of dataset, changes will have to occur there and maybe here for dataset layout information. (h5stat.c 1.14.4 L451)
- `static herr_t print_dataset_info(const iter_t *iter)`
  - Update to accommodate sparse chunk data. Currently, it only has a case for chunked, contiguous, compact, and virtual. (`h5stat.c 1.14.4 L1276`)
  - Accommodate how filters are applied to different sections of sparse chunk datasets. (h5stat.c 1.14.4 L1289)

1.3.4 Testing

Testing should be added to make sure that h5stat correctly outputs information for datasets using the new structured chunk storage layout.

1.4 h5import

1.4.1 Output example

Old ascii file support should be removed, and this tool should only rely on the new h5dump ddl data files. There will need to be changes to the help text to reflect the addition of structured chunks.

The tool is interoperable with h5dump, and we will need to preserve this option for sparse data when considering updates to configuration file and input data file. I.e., it would be beneficial to co-design h5dump and h5import changes.

1.4.2 New flags and options

h5import will not need any new flags or options.

1.4.3 Required code changes

In `h5import.c`, `static int processConfigurationFile(char *infile, struct Input *in)` must be updated for structured chunk. (h5import.c 1.14.4 L1816-1986)

1.5 h5repack

`h5repack` should be updated to repack a dataset from/to structured chunk storage layout.

1.5.1 Output example

```
% h5repack -l Sparse:SPARSECHUNK=5x4 BLOCK  (2,2)-(4,7), (6,0)-(6,2) POINT
(5,9), (11, 1), (12,8) dense.h5 sparse.h5
```

This approach may not work for huge datasets and some automation for creating `BLOCK/POINT` input list or automated detection of "defined value" would be required. For example, `h5dump` can be updated to print locations in the required `BLOCK/POINT` format only for the values that are not equal to some specified value, or `h5repack` may scan the data and exclude the elements that have a specified value. For example,

```
% h5repack -l Sparse:SPARSECHUNK=5x4 –exclude 0 dense.h5 sparse.h5
```

Such approach is problematic as shown by the following example. In our original example of sparse matrix stored using sparse chunk layout, the second element in the `BLOCK (6,0) – (6,2)` has value 0 and it is defined. If we repack the file to dense layout the default library fill value 0 will be used for undefined values. If we repack back to the sparse chunk layout using "`–exclude 0`", the file will be different since the original `BLOCK (6,0)-(6,2)` will become `POINT(6,0), (6,2),` i.e. we will violate round trip behavior for `h5repack`.

1.5.2 New flags and options

Existing flags can be used to change storage from sparse chunk to the chunked and to the contiguous storage layouts.

We will need to introduce a new flag, e.g., "`SPARSECHUNK`", when repacking to use sparse chunk storage. Please note that without specifying defined values, sparse chunk storage will not achieve space saving functionality since all data elements of each chunk will be stored along with the dataspace information for the chunk. In order to take advantage of sparse chunk storage, we will need to introduce new flag, `--defined-elements`, followed by the list of `BLOCK`s and `POINT`s using the format as specified by the `h5dump` tool's DDL as shown above. We assume that example matrix shown in Figure 3 is stored as a regular HDF5 dataset and is repacked to use sparse chunk storage layout using new option and known locations for defined elements.

1.5.3 Required code changes

In regard to the API changes that must be made, in `h5repack.h`, we should add to the Usage Example section at the bottom of help/Doxygen text of how to convert from and to the new structured chunk storage layout type.

In h5repack.c

- Does h5repack_addfilter requires no changing for how filters are applied to structured chunk, but sections may have their own filter pipelines, so the flag `--defined-elements` will have to be used to specify which section.
- int h5repack_addlayout(const char *str, pack_opt_t *options) should be updated to add in structured chunk as an option (1.14.4 h5repack.c L179).
- static int check_options(pack_opt_t *options) should add a case for structured chunk (1.14.4 h5repack.c L619).

In h5repack_copy.c

- ~~int get_hyperslab(hid_t dcpl_id, int rank_dset, const hsize_t dims_dset[], size_t size_datum, hsize_t dims_hslab[], hsize_t *hslab_nbytes_p) should be updated with a new case for structured chunk (1.14.4 h5repack_copy.c L453). This depends on how data is used. If there is the option for defined selection, then it must be applied when we write to dataset. If it is a get, then no changes are needed, but some might be needed with a write.~~
- int do_copy_objects(hid_t fidin, hid_t fidout, trav_table_t *travt, pack_opt_t *options) should add option for converting to new structured chunk storage layout type. (1.14.4 h5repack_copy.c L977)

In h5repack_filters.c

- int apply_filters() needs to be updated for structured chunk (1.14.4 h5repack_filters.c L303). Specifically, this section needs to check for structured chunk and if so get each section and change how it handles filters.
- static int aux_assign_obj() needs an additional case for structured chunk that changes how filters are applied (1.14.4 h5repack_filters.c L140).
- static int aux_copy_obj() should also add an if statement for handling structured chunk (1.14.4 h5repack_filters.c L66).

In h5repack_main.c

- For the static void usage(const char *prog) method (1.14.4 h5repack_main.c L99), we must add structured chunk into the usage text.

In 5repack_opttable.c

- static void aux_tblinsert_filter()
  - We will need a new option for structured chunk in the table (1.14.4 h5repack_opttable.c L55)

In h5repack_parse.c

- obj_list_t *parse_layout(), we must add structured chunk as a storage layout here (1.14.4 h5repack_parse.c L549). It looks like this section is also missing virtual. These things could be updated at the same time.

In h5repack_verify.c

- int verify_layout(hid_t pid, pack_info_t *obj), as above, we must add structured chunk as a storage layout (1.14.4 h5repack_verify.c L323) and also look into updating for virtual.

1.6 h5diff

Because the tool doesn't compare layouts when diffing two datasets, no changes to the flags or the tool's output will be required at this time. Currently, h5diff only gets storage layout to check storage size.

Some potential future work would include adding a compare for selections of structured chunk. In that case, we would diff the SECTIONS for any selection differences. Further discussion is also required to add comparison of selection data, and we will investigate adding better comparison logic for comparing stored defined elements.

2 Tools that do not require updating

2.1 h5debug

h5debug will itself will likely not require updates, but it will need the support of the debugging callbacks in the library for structured chunk related changes. Specifically, H5FAdbg.c and H5EAdbg.c should be investigated as places to change.

2.2 h5copy

h5copy does not require storage layout when copying and will use existing functionality to copy sparse datasets. Therefore, it will not require updating. This tool uses H5Ocopy to do its work, so H5Ocopy must be checked for necessary updates for structured chunk. It is recommended to look at H5O__copy.

2.3 h5format_convert

Nothing is currently planned for h5format_convert. The only thing to change would be to update the help text to display that no action will be taken for structured chunk datasets similarly to virtual datasets.

2.4 h5jam

h5jam adds a new superblock to the front of an HDF5 file and concatenates it. This tool currently does not need any changes to function after structured chunk is introduced since it should not impact how this tool works at all.

2.5 h5clear

h5clear allows the user to clear the superblock status flag field, remove the metadata cache image, print the EOA and EOF, or set the EOA of a file. None of these functions require any updates because of structured chunk, so no changes are necessary.

2.6 h5perf

h5perf is a parallel file system benchmark tool, so no changes are necessary.

2.7 h5delete

h5delete is a tool to delete HDF5 files, so no changes are necessary.

2.8 h5mkgrp

h5mkgrp is a tool that creates groups in an HDF5 file. Because this does not need to interact with structured chunk, nothing needs to be changed.

2.9 h5repart

h5repart repartitions a file family. This also doesn't require any changes with the addition of structured chunk.

Revision History

October 24, 2024	Version 2 is a formatted version of Version 1 sent on 9/14/2024 to Lifeboat.