Using the Direct Chunk Write Function

Introduced with

HDF5 Release 1.8.11

28 February 2013

The HDF Group

http://www.HDFGroup.org

Copyright Notice and License Terms for HDF5 (Hierarchical Data Format 5) Software Library and Utilities

Contents

1. Using the Direct Chunk Write Function

When a user application has a chunked dataset and is trying to write a single chunk of data with `H5Dwrite`, the data goes through several steps inside the HDF5 library. The library first examines the hyperslab selection. Then it converts the data from the datatype in memory to the datatype in the file if they are different. Finally, the library processes the data in the filter pipeline. Starting with the 1.8.11 release, a new high-level C function called `H5DOwrite_chunk` becomes available. It writes a data chunk directly to the file bypassing the library's hyperslab selection, data conversion, and filter pipeline processes. In other words, if an application can pre-process the data, then the application can use `H5DOwrite_chunk` to write the data much faster.

`H5DOwrite_chunk` was developed in response to a client request. The client builds X-ray pixel detectors for use at synchrotron light sources. These detectors can produce data at the rate of tens of gigabytes per second. Before transferring the data over their network, the detectors compress the data by a factor of 10 or more. The modular architecture of the detectors can scale up its data stream in parallel and maps well to current parallel computing and storage systems.

1.1. Using the H5DOwrite_chunk Function

Basically, the `H5DOwrite_chunk` function takes a pre-processed data chunk (`buf`) and its size (`data_size`) and writes to the chunk location (`offset`) in the dataset (`dset_id`).

The function prototype is shown below:

```
herr_t H5DOwrite_chunk(
       hid_t         dset_id,     /*the dataset                        */
       hid_t         dxpl_id,     /*data transfer property list        */
       uint32_t      filter_mask, /*indicates which filters are used */
       hsize_t *     offset,      /*position of the chunk              */
       size_t        data_size,   /*size of the actual data            */
       const void *  buf          /*buffer with data to be written     */
       )
```

Below is a simple example showing how to use the function:

```
hsize_t offset[2] = {4, 4};
uint32_t filter_mask = 0;
size_t nbytes = 40;

if(H5DOwrite_chunk(dset_id, dxpl, filter_mask,
       offset, nbytes, data_buf) < 0)
       goto error;
```

Example 1. Using H5DOwrite_chunk

In the example above, the dataset is 8x8 elements of `int`. Each chunk is 4x4. The offset of the first element of the chunk to be written is 4 and 4. In the diagram below, the shaded chunk is the data to be

written. The function is writing a pre-compressed data chunk of 40 bytes (assumed) to the dataset. The zero value of the filter mask means that all filters have been applied to the pre-processed data.

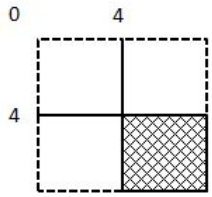

Figure 1. Illustration of the chunk to be written in the example code above

The complete code example at the end of this topic shows how to set the value of the filter mask to indicate a filter being skipped. The corresponding bit in the filter mask is turned on when a filter is skipped. For example, if the second filter is skipped, the second bit of the filter mask should be turned on. For more information, see the `H5DOwrite_chunk` entry in the *HDF5 Reference Manual*.

1.2. The Design

The following diagram shows how the function `H5DOwrite_chunk` bypasses hyperslab selection, data conversion, and filter pipeline inside the HDF5 library.

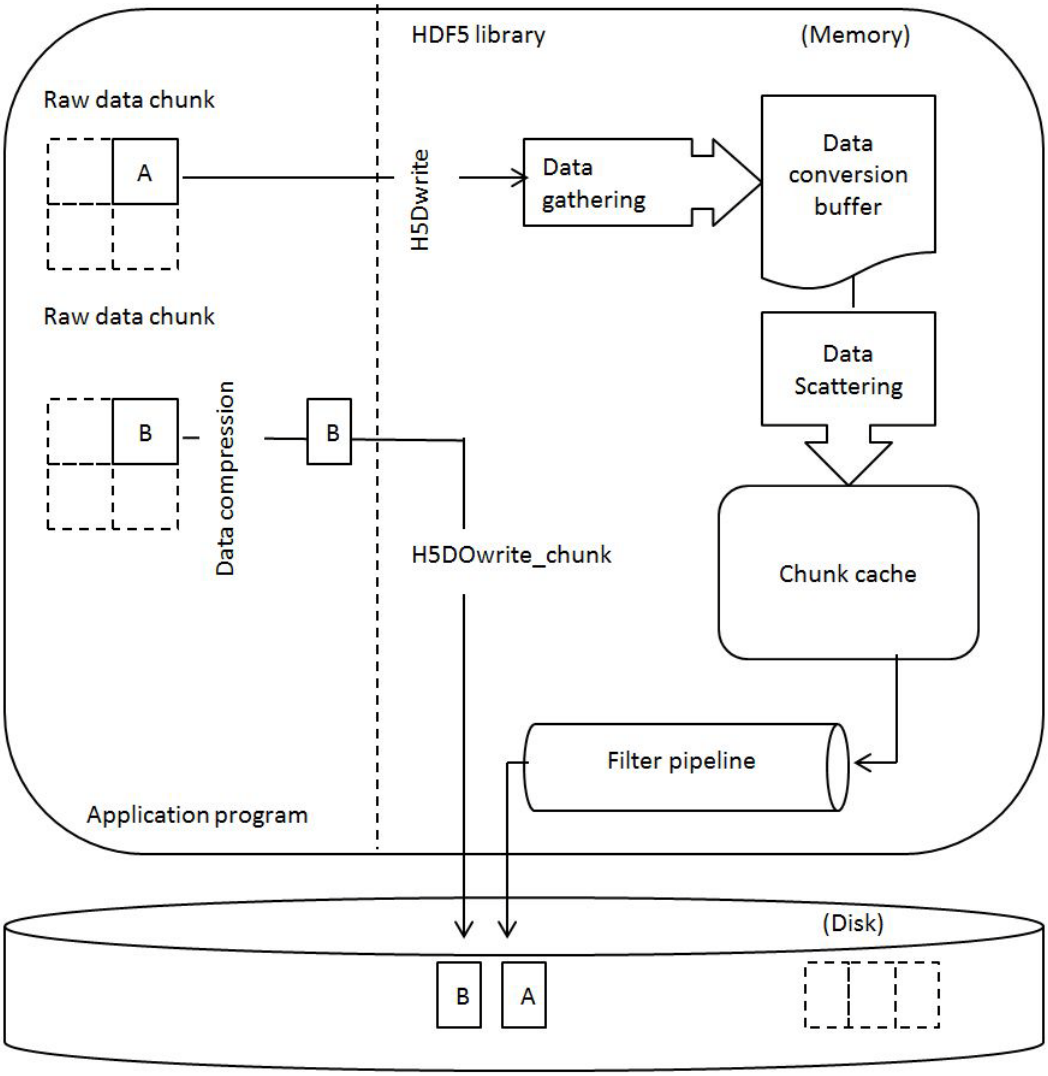

Figure 2. Diagram for `H5DOwrite_chunk` in the HDF5 Library

1.3. Performance

The table below describes the results of performance benchmark tests run by HDF developers. It shows that using the new function `H5DOwrite_chunk` to write pre-compressed data is much faster than using the `H5Dwrite` function to compress and write the same data with the filter pipeline. Measurements involving `H5Dwrite` include compression time in the filter pipeline. Since the data is already compressed before `H5DOwrite_chunk` is called, use of `H5DOwrite_chunk` to write compressed data avoids the performance bottleneck in the HDF5 filter pipeline.

The test was run on a Linux 2.6.18 / 64-bit Intel x86_64 machine. The dataset contained 100 chunks. Only one chunk was written to the file per write call. The number of writes was 100. The time measurement was for the entire dataset with the Unix system function `gettimeofday`. Writing the entire dataset with one write call took almost the same amount of time as writing chunk by chunk. In order to force the system to flush the data to the file, the `O_SYNC` flag was used to open the file.

Table 1. Performance result for `H5DOwrite_chunk` in the high-level library

Dataset size (MB)	95.37		762.94		2288.82	
Size after compression (MB)	64.14		512.94		1538.81	
Dataset dimensionality	100x1000x250		100x2000x1000		100x2000x3000	
Chunk dimensionality	1000x250		2000x1000		2000x3000	
Datatype	4-byte integer		4-byte integer		4-byte integer	
	speed[1]	time[2]	speed	time	speed	time
`H5Dwrite` writes without compression filter	77.27	1.23	97.02	7.86	91.77	24.94
`H5DOwrite_chunk` writes uncompressed data	79	1.21	95.71	7.97	89.17	25.67
`H5Dwrite` writes with compression filter	**2.68**	**35.59**	**2.67**	**285.75**	**2.67**	**857.24**
`H5DOwrite_chunk` writes compressed data	**77.19**	**0.83**	**78.56**	**6.53**	**96.28**	**15.98**
Unix writes compressed data to Unix file	76.49	0.84	95	5.4	98.59	15.61

[1] IO speed is in MB/s.
[2] Time is in second (s).

1.4. A Word of Caution

Since `H5DOwrite_chunk` writes data chunks directly in a file, developers must be careful when using it. The function bypasses hyperslab selection, the conversion of data from one datatype to another, and the filter pipeline to write the chunk. Developers should have experience with these processes before they use this function.

1.5. A Complete Code Example

The following is an example of using `H5DOwrite_chunk` to write an entire dataset by chunk.

```
#include <zlib.h>
#include <math.h>
#define DEFLATE_SIZE_ADJUST(s) (ceil(((double)(s))*1.001)+12)
             :
size_t       buf_size = CHUNK_NX*CHUNK_NY*sizeof(int);
const Bytef *z_src = (const Bytef*)(direct_buf);
Bytef       *z_dst;             /*destination buffer            */
uLongf       z_dst_nbytes = (uLongf)DEFLATE_SIZE_ADJUST(buf_size);
uLong        z_src_nbytes = (uLong)buf_size;
int          aggression = 9;     /* Compression aggression setting */
uint32_t     filter_mask = 0;
size_t       buf_size = CHUNK_NX*CHUNK_NY*sizeof(int);

/* Create the data space */
if((dataspace = H5Screate_simple(RANK, dims, maxdims)) < 0)
      goto error;

/* Create a new file */
if((file = H5Fcreate(FILE_NAME5, H5F_ACC_TRUNC, H5P_DEFAULT,
             H5P_DEFAULT)) < 0)
      goto error;

/* Modify dataset creation properties, i.e. enable chunking
   and compression */
if((cparms = H5Pcreate(H5P_DATASET_CREATE)) < 0)
      goto error;

if((status = H5Pset_chunk( cparms, RANK, chunk_dims)) < 0)
      goto error;

if((status = H5Pset_deflate( cparms, aggression)) < 0)
      goto error;

/* Create a new dataset within the file using cparms creation
   properties */
if((dset_id = H5Dcreate2(file, DATASETNAME, H5T_NATIVE_INT, dataspace,
                    H5P_DEFAULT,cparms, H5P_DEFAULT)) < 0)
       goto error;

/* Initialize data for one chunk */
for(i = n = 0; i < CHUNK_NX; i++)
    for(j = 0; j < CHUNK_NY; j++)
        direct_buf[i][j] = n++;

/* Allocate output (compressed) buffer */
outbuf = malloc(z_dst_nbytes);
z_dst = (Bytef *)outbuf;
```

```
/* Perform compression from the source to the destination buffer */
ret = compress2(z_dst, &z_dst_nbytes, z_src, z_src_nbytes, aggression);

/* Check for various zlib errors */
if(Z_BUF_ERROR == ret) {
       fprintf(stderr, "overflow");
       goto error;
} else if(Z_MEM_ERROR == ret) {
       fprintf(stderr, "deflate memory error");
       goto error;
} else if(Z_OK != ret) {
       fprintf(stderr, "other deflate error");
       goto error;
}

/* Write the compressed chunk data repeatedly to cover all the chunks in
 * the dataset, using the direct write function.     */
for(i=0; i<NX/CHUNK_NX; i++) {
       for(j=0; j<NY/CHUNK_NY; j++) {
              status = H5DOwrite_chunk(dset_id, H5P_DEFAULT,
                     filter_mask, offset, z_dst_nbytes, outbuf);
              offset[1] += CHUNK_NY;
       }
       offset[0] += CHUNK_NX;
       offset[1] = 0;
}

/* Overwrite the first chunk with uncompressed data.  Set the filter
 * mask to indicate the compression filter is skipped */
filter_mask = 0x00000001;
offset[0] = offset[1] = 0;
if(H5DOwrite_chunk(dset_id, H5P_DEFAULT, filter_mask, offset, buf_size,
                     direct_buf) < 0)
       goto error;

/* Read the entire dataset back for data verification converting ints
 * to longs*/
if(H5Dread(dataset, H5T_NATIVE_LONG, H5S_ALL, H5S_ALL, H5P_DEFAULT,
              outbuf_long) < 0)
       goto error;

/* Data verification here */
              :
              :
```

Example 2. A complete code example for H5DOwrite chunk