Bypass VOL Connector

Current Implementation as of the end of January, 2025

Bypass VOL

Before the multi-thread support for all the components of the library is fully implemented, the work for Bypass VOL is being done in several stages. Because the current implementation of the HDF5 library's multi-thread feature is to serialize multiple threads into a sequence of threads (threads are executed one by one), this stage of Bypass VOL uses a thread pool to process data reading.

Bypass VOL is mostly based on the External Pass-through VOL (https://github.com/hpc-io/vol-external-passthrough).

Limitation

In this stage, Bypass VOL only handles a few limited cases for reading datasets. For the following cases, it behaves the same way as External Pass-through VOL, where the I/O activity is simply passed down to the underneath VOL (normally the native VOL):

- data conversion (the datatypes in the file and memory are not the same)
- filters
- virtual datasets (VDS)
- external file
- external link
- certain datatypes, including reference (object and region), variable-length, time, opaque, compound, array

The Bypass VOL doesn't issue any warning or error when it encounters the cases above. We plan to address these limitations as the project continues.

Thread pool

The Bypass VOL connector uses a thread pool to handle multiple read tasks. Thread pool is a common practice of multi-thread computation. Basically, the main process puts a sequence of tasks into a queue and signal the thread pool to start. Each thread in the thread pool grabs a number (default is 1024) of tasks to read. When there is no task in the queue, all threads in the pool wait until the main process puts another round of tasks into the queue and sends out a signal.

The thread pool is initialized in the function `H5VL_bypass_init()` and terminated in the function `H5VL_bypass_term()`. The default number of threads is 4. But the application can change it through the environment variable `BYPASS_VOL_NTHREADS`.

Chunked Datasets

When the application reads a hyperslab selection of a chunked dataset, the callback function `H5VL_bypass_dataset_read()` of Bypass VOL (located in `H5VLbypass.c`) uses the following algorithm:

1. It iterates through all the chunks in the dataset and finds the intersection of a particular chunk and the data selection in the dataset. For each chunk containing the data selection, the Bypass VOL figures out where the data should go in the application's read buffer using the function `H5Sselect_project_intersection()`.
2. This pair of dataspace selections (the chunk containing data selection in the file and the corresponding selection in the memory) are used to create two respective selection iterators with the function `H5Ssel_iter_create()`.
3. The Bypass VOL obtains the offset and length of each data piece in file and in memory from the two iterators using the function `H5Ssel_iter_get_seq_list()`. If the length of the data piece is greater than the default size (1 MB), this data piece is chopped down to more than one pieces. The application can adjust the length of the data piece through the environment variable `BYPASS_VOL_IO_SIZE` (to be implemented).
4. It puts this necessary information (location of the chunk, the offset and length of data in the chunk and in memory) into the queue for the thread pool and send a signal to the threads in the thread pool. After receiving this signal, the available threads in the pool grab the information (tasks) and start to read the data.
5. Then it moves to the next chunk and repeats Step 1 – 4.

Contiguous Datasets

When the application reads a contiguous dataset, the callback function `H5VL_bypass_dataset_read()` of Bypass VOL basically uses the same algorithm as the chunked dataset. It treats the entire contiguous dataset as a single chunk.

Reading multiple datasets

If the application tries to read multiple datasets using `H5Dread_multi()`, the the callback function `H5VL_bypass_dataset_read()` of Bypass VOL simply loops through those datasets and process them one by one. It can handle the datasets from difference files, while the HDF5 library's native `H5Dread_multi()` can only handle the datasets from the same file.

Dataspace selection for data reading

The implementation of data selection in reading in the Bypass VOL follows the native `H5Dread()` and `H5Dread_multi()`. The reference manual entry of `H5Dread()` gives the following guideline:

mem_space_id	**file_space_id**	**Behavior**
valid dataspace ID	valid dataspace ID	mem_space_id specifies the memory dataspace and the selection within it. file_space_id specifies the selection within the file dataset's dataspace.
H5S_ALL	valid dataspace ID	The file dataset's dataspace is used for the memory dataspace and the selection specified with file_space_id specifies the selection within it. The combination of the file dataset's dataspace and the selection from file_space_id is used for memory also.

valid dataspace ID	**H5S_ALL**	mem_space_id specifies the memory dataspace and the selection within it. The selection within the file dataset's dataspace is set to the "all" selection.
H5S_ALL	**H5S_ALL**	The file dataset's dataspace is used for the memory dataspace and the selection within the memory dataspace is set to the "all" selection. The selection within the file dataset's dataspace is set to the "all" selection.

Other Implementation Details

`pread()` also needs the file descriptor (an integer type) as a parameter. To find the right file descriptor, Bypass VOL maintains a table containing the file descriptors of all the opened files. These file descriptors are obtained during the file open stage in `H5VL_bypass_file_open()`. To help locate the file descriptor, the table also contains the matching file names. The file descriptor is closed during the call back function `H5VL_bypass_file_close()`.

`pread()` has a data limit of 2GB – 1. The current implementation returns an error if the data is equal to or greater than 2GB.

To reduce the number of querying datasets' information in `H5VL_bypass_dataset_read()`, Bypass VOL also maintains a table containing the information of datasets being opened, such as the names, file names, layout types, data type IDs, and location. These pieces of information are obtained during `H5VL_bypass_dataset_open()`. And released during `H5VL_bypass_dataset_close()`.

The table containing the file descriptors of opened files is defined in `H5VLbypass_private.h`. The structure is called `file_t`. The table containing the information of opened datasets is called `dset_t`, which is also in `H5VLbypass_private.h`. These two tables are dynamically allocated. Once they are filled up, the number of entries will double.

Next Step

Writing to chunked datasets is supposed to be similar to read. Bypass VOL must figure out which chunks are supposed to be written and their locations in the file and in the application's buffer. The limitation is the same as reading: no datatype conversion and no filter. The file name, location of the dataset, and the hyperslab selection are necessary information to write data. As reading datasets has demonstrated, Bypass VOL is able to obtain these pieces of information. Writing to contiguous datasets can still be treated as a special case of chunked datasets, where the whole dataset is simply one chunk.

Performance Benchmark Program

In order to compare the performance of multi-threaded data reading, the benchmark program (`h5_read.c`) uses the following three methods:

1. Using the HDF5 library built with multi-thread: As mentioned above, the current implementation of the multi-thread option of the HDF5 library serializes multiple threads into a sequence of threads. This limitation actually slows down the speed of multi-threaded read. For comparison purpose, this method is only run as a serial program.

2. Using the Bypass VOL with the HDF5 library built with multi-thread: Although the HDF5 library does not currently support multi-threaded read, the Bypass VOL internally uses a thread pool to read the data. It bypasses the HDF5 library during the I/O activities. At the same time, it outputs the necessary information (the file name, dataset name, chunk or dataset location in a HDF5 file, the offset of the data in the file to be read from, the number of elements, and the offset of the data in memory to be read into) to a log file. This log file is for the third method below to use.

3. Using only POSIX functions: Taking the log file generated from the second method above, this method opens, reads, and closes the data file using a thread pool with POSIX functions. Multi-threaded read can be done in parallel with this method.

Each method above reads data in four scenarios:

1. Reading a single dataset;
2. Reading multiple datasets;
3. Reading multiple files with each file having a single dataset;
4. Reading multiple datasets in multiple files (to be implemented).