h5bench

Release 1.2

Suren Byna, Houjun Tang, Quincey Koziol, Tony Li, John Ravi, Sc

Jun 30, 2023

GETTING STARTED

h5bench is a suite of parallel I/O benchmarks or kernels representing I/O patterns that are commonly used in HDF5 applications on high performance computing systems. H5bench measures I/O performance from various aspects, including the I/O overhead, observed I/O rate, etc.

These are the benchmarks and kernels currently available in h5bench:

Benchmark	Name	SYNC	ASYNC VOL
h5bench write	`h5bench_write`		
h5bench read	`h5bench_read`		
GPU-IO (CUDA)	`h5bench_read_cuda` `h5bench_write_cuda`		
Metadata Stress	`h5bench_hdf5_iotest`		
AMReX	`h5bench_amrex`		
Exerciser	`h5bench_exerciser`		
OpenPMD (write)	`h5bench_openpmd_write`		
OpenPMD (read)	`h5bench_openpmd_read`		
E3SM-IO	`h5bench_e3sm`		

CHAPTER

ONE

BUILD INSTRUCTIONS

1.1 Build with CMake (recommended)

First, clone the h5bench GitHub repository and ensure you are cloning the submodules:

```
git clone --recurse-submodules https://github.com/hpc-io/h5bench
```

If you are upadting your h5bench, ensure you have the latest submodules that could be included in new releases:

```
git submodule update --init
```

1.1.1 Dependency and environment variable settings

H5bench depends on MPI and parallel HDF5.

Use system provided by HDF5

For instance on the Cori system at NERSC:

```
module load cray-hdf5-parallel
```

You can also load any paralel HDF5 provided on your system, and you are good to go.

Use your own installed HDF5

Make sure to unload any system provided HDF5 version:, and set an environment variable to specify the HDF5 install path:

```
export HDF5_HOME=/path/to/your/hdf5/installation
```

It should point to a path that contains the `include/`, `lib/`, and `bin/` subdirectories.

Enable with GPU transfers with CUDA

You can test GPU memory transfers bandwidth if you compile with CUDA support.

For instance on Summit system at OLCF:

```
module load cuda
```

1.1.2 Compile with CMake

In the source directory of your cloned h5bench repository, run the following:

```
mkdir build
cd build

cmake ..

make
make install
```

By default, h5bench will only compile the base write and read benchmarks. To enable the additional benchmarks, you need to explicitly enable them before building h5bench. You can also enable all the benchmarks with `-DH5BENCH_ALL=ON`. Notice that some of them have additional dependencies.

Benchmark	Name	Build
h5bench write	`h5bench_write`	Always
h5bench read	`h5bench_read`	Always
Metadata Stress	`h5bench_hdf5_iotest`	`-DH5BENCH_METADATA=ON`
AMReX	`h5bench_amrex`	`-DH5BENCH_AMREX=ON`
Exerciser	`h5bench_exerciser`	`-DH5BENCH_EXERCISER=ON`
OpenPMD (write)	`h5bench_openpmd_write`	`-DH5BENCH_OPENPMD=ON`
OpenPMD (read)	`h5bench_openpmd_read`	`-DH5BENCH_OPENPMD=ON`
E3SM-IO	`h5bench_e3sm`	`-DH5BENCH_E3SM=ON`
GPU-IO (cuda)	`h5bench_cuda_read` `h5bench_cuda_write`	`-DH5BENCH_CUDA=ON`

Warning: If you want to specify the installation directory, you can pass `-DCMAKE_INSTALL_PREFIX` to `cmake`. If you are not installing it, make sure when you run `h5bench`, you update your environment variables to include the *build* directory. Otherwise, h5bench will not be able to find all the benchmarks.

1.1.3 Build with HDF5 ASYNC VOL connector support

To run `_async` benchmarks, you need the develop branch of **both** HDF5 and ASYNC-VOL. When building h5bench you need to specify the `-DWITH_ASYNC_VOL:BOOL=ON` option and have already compiled the VOL connector in the `$ASYNC_VOL` directory:

```
mkdir build
cd build

cmake .. -DWITH_ASYNC_VOL=ON -DCMAKE_C_FLAGS="-I/$ASYNC_VOL/src -L/$ASYNC_VOL/src"
```

(continues on next page)

(continued from previous page)

```
make
make install
```

h5bench will automatically set the environment variables required to run the asynchronous versions, as long as you specify them in your JSON configuration file. However, if you run the benchmarks manually, you will need to set the following environment variables:

```
export HDF5_HOME="$YOUR_HDF5_DEVELOP_BRANCH_BUILD/hdf5"
export ASYNC_HOME="$YOUR_ASYNC_VOL/src"

export HDF5_VOL_CONNECTOR="async under_vol=0;under_info={}"
export HDF5_PLUGIN_PATH="$ASYNC_HOME"

# Linux
export LD_LIBRARY_PATH="$HDF5_HOME/lib:$ASYNC_HOME"
# MacOS
export DYLD_LIBRARY_PATH="$HDF5_HOME/lib:$ASYNC_HOME"
```

1.2 Build with Spack

You can also use Spack to install h5bench:

```
spack install h5bench
```

There are some variants available as described bellow:

```
CMakePackage:   h5bench

Description:
    A benchmark suite for measuring HDF5 performance.

Homepage: https://github.com/hpc-io/h5bench

Preferred version:
    1.2         [git] https://github.com/hpc-io/h5bench.git at commit␣
↪866af6777573d20740d02acc47a9080de093e4ad

Safe versions:
    develop     [git] https://github.com/hpc-io/h5bench.git on branch develop
    1.2         [git] https://github.com/hpc-io/h5bench.git at commit␣
↪866af6777573d20740d02acc47a9080de093e4ad
    1.1         [git] https://github.com/hpc-io/h5bench.git at commit␣
↪1276530a128025b83a4d9e3814a98f92876bb5c4
    1.0         [git] https://github.com/hpc-io/h5bench.git at commit␣
↪9d3438c1bc66c5976279ef203bd11a8d48ade724
    latest      [git] https://github.com/hpc-io/h5bench.git on branch master

Deprecated versions:
    None
```

(continues on next page)

(continued from previous page)

```
Variants:
    Name [Default]                  When       Allowed values          Description
    ============================    =======    ====================    ␣
↪================================

    all [off]                       @1.2:      on, off                 Enables all␣
↪h5bench benchmarks
    amrex [off]                     @1.2:      on, off                 Enables AMReX␣
↪benchmark
    build_type [RelWithDebInfo]     --         Debug, Release,         CMake build type
                                               RelWithDebInfo,
                                               MinSizeRel
    e3sm [off]                      @1.2:      on, off                 Enables E3SM␣
↪benchmark
    exerciser [off]                 @1.2:      on, off                 Enables exerciser␣
↪benchmark
    ipo [off]                       --         on, off                 CMake␣
↪interprocedural optimization
    metadata [off]                  @1.2:      on, off                 Enables metadata␣
↪benchmark
    openpmd [off]                   @1.2:      on, off                 Enables OpenPMD␣
↪benchmark

Build Dependencies:
    cmake  hdf5  mpi  parallel-netcdf

Link Dependencies:
    hdf5  mpi  parallel-netcdf

Run Dependencies:
    None
```

Warning: Current h5bench versions in Spack do not have support for the HDF5 VOL async/cache connectors yet.

CHAPTER

TWO

RUNNING H5BENCH

2.1 h5bench (recommended)

We provide a single script you can use to run the benchmarks available in the h5bench Benchmarking Suite. You can combine multiple benchmarks into a workflow with distinct configurations. If you prefer, you can also manually run each benchmark in h5bench. For more details, refer to the Manual Execution section.

```
usage: h5bench [-h] [--abort-on-failure] [--debug] [--validate-mode] setup

H5bench: a Parallel I/O Benchmark Suite for HDF5:

positional arguments:
  setup               JSON file with the benchmarks to run

optional arguments:
  -h, --help          show this help message and exit
  --abort-on-failure  Stop h5bench if a benchmark failed
  --debug             Enable debug mode
  --validate-mode     Validated if the requested mode (async/sync) was run
```

You need to provide a JSON file with the configurations you want to run. If you're using `h5bench`, you should *not* call `mpirun`, `srun`, or any other parallel launcher on your own. Refer to the manual execution section if you want to follow that approach instead. The main script will handle setting and unsetting environment variables, launching the benchmarks with the provided configuration and HDF5 VOL connectors.

```
./h5bench configuration.json
```

If you run it with the `--debug` option, h5bench will also print log messages `stdout`. The default behavior is to store it in a file.

Warning: Make sure you do not call *srun*, *mpirun*, etc directly but instead define that in the JSON configuration file. You should **always** call h5bench directly.

2.1.1 Configuration

The JSON configuration file has five main properties: `mpi`, `vol`, `file-system`, `directory`, `benchmarks`. All should be defined, even if empty.

MPI

You can set the MPI launcher you want to use, e.g. `mpirun`, `mpiexec`, and `srun`, and provide the number of processes you want to use. For other methods or a fine grain control on the job configuration, you can define the `configuration` properties that h5bench will use to launch the experiments using the `command` property you provided. If the `configuration` option is defined, h5bench will ignore the `ranks` property.

```
"mpi": {
   "command": "mpirun",
   "ranks": "4",
   "configuration": "-np 8 --oversubscribe"
}
```

VOL

You can use HDF5 VOL connectors (async, cache, etc) for `h5bench_write` and `h5bench_read`. Because some benchmarks inside h5bench do not have support for VOL connectors yet, you need to provide the necessary information in the configuration file to handle the VOL setup during runtime.

```
"vol": {
   "library": "/vol-async/src:/hdf5-async-vol-register-install/lib:/argobots/install/
↪lib:/hdf5-install/install:",
   "path": "/vol-async/src",
   "connector": "async under_vol=0;under_info={}"
}
```

You should provide the absolute path for all the libraries required by the VOL connector using the `library` property, the `path` of the VOL connector, and the configuration in `connector`. The provided example depicts how to configure the HDF5 VOL async connector.

Directory

h5bench will create a directory for the given execution workflow, where it will store all the generated files and logs. Additional options such as data striping for Lustre, if configured, will be applied to this directory.

```
"directory": "hdf5-output-directory"
```

File System

You can use this property to configure some file system options. For now, you can use it for Lustre to define the striping count and size that should be applied to the `directory` that will store all the generated data from `h5bench`.

```
"file-system": {
   "lustre": {
      "stripe-size": "1M",
      "stripe-count": "4"
   }
}
```

Benchmarks

You can specify which benchmarks `h5bench` should run using this property, their order, and configuration. You can choose between: `write`, `write-unlimited`, `overwrite`, `append`, `read`, `metadata`, `exerciser`, `openpmd`, `amrex`, `e3sm`.

For each pattern of `h5bench`, you should provide the `file` and the `configuration`:

```
{
   "benchmark": "write",
   "file": "test.h5",
   "configuration": {
      "MEM_PATTERN": "CONTIG",
      "FILE_PATTERN": "CONTIG",
      "NUM_PARTICLES": "16 M",
      "TIMESTEPS": "5",
      "DELAYED_CLOSE_TIMESTEPS": "2",
      "COLLECTIVE_DATA": "NO",
      "COLLECTIVE_METADATA": "NO",
      "EMULATED_COMPUTE_TIME_PER_TIMESTEP": "1 s",
      "NUM_DIMS": "1",
      "DIM_1": "16777216",
      "DIM_2": "1",
      "DIM_3": "1",
      "MODE": "SYNC",
      "CSV_FILE": "output.csv"
   }
}
```

If you provide the same `file` name used for a previous `write` execution, it will read from that file. This way, you can configure a workflow with multiple interleaving files, e.g., `write` file-01, `write` file-02, `read` file-02, `read` file-01.

```
{
   "benchmark": "read": {
   "file": "test.h5",
   "configuration": {
      "MEM_PATTERN": "CONTIG",
      "FILE_PATTERN": "CONTIG",
      "NUM_PARTICLES": "16 M",
      "TIMESTEPS": "5",
      "DELAYED_CLOSE_TIMESTEPS": "2",
```

(continues on next page)

(continued from previous page)

```
      "COLLECTIVE_DATA": "NO",
      "COLLECTIVE_METADATA": "NO",
      "EMULATED_COMPUTE_TIME_PER_TIMESTEP": "1 s",
      "NUM_DIMS": "1",
      "DIM_1": "16777216",
      "DIM_2": "1",
      "DIM_3": "1",
      "MODE": "SYNC",
      "CSV_FILE": "output.csv"
   }
}
```

For the `metadata` stress benchmark, `file` and `configuration` properties must be defined:

```
{
   "benchmark": "metadata",
   "file": "hdf5_iotest.h5",
   "configuration": {
      "version": "0",
      "steps": "20",
      "arrays": "500",
      "rows": "100",
      "columns": "200",
      "process-rows": "2",
      "process-columns": "2",
      "scaling": "weak",
      "dataset-rank": "4",
      "slowest-dimension": "step",
      "layout": "contiguous",
      "mpi-io": "independent",
      "csv-file": "hdf5_iotest.csv"
   }
}
```

For the `exerciser` benchmark, you need to provide the required runtime options in the JSON file inside the `configuration` property.

```
{
   "benchmark": "exerciser",
   "configuration": {
      "numdims": "2",
      "minels": "8 8",
      "nsizes": "3",
      "bufmult": "2 2",
      "dimranks": "8 4"
   }
}
```

You can find several samples of configuration file with all the optins in the our [GitHub repository] (https://github.com/hpc-io/h5bench/tree/master/samples). You can also refer to this sample of a complete `configuration.json` file that defined the workflow of the execution of multiple benchmarks from h5bench Suite:

```
{
    "mpi": {
        "command": "mpirun",
        "ranks": "4",
        "configuration": "-np 8 --oversubscribe"
    },
    "vol": {
        "library": "/vol-async/src:/hdf5-async-vol-register-install/lib:/argobots/
↪install/lib:/hdf5-install/install:",
        "path": "/vol-async/src",
        "connector": "async under_vol=0;under_info={}"
    },
    "file-system": {
        "lustre": {
            "stripe-size": "1M",
            "stripe-count": "4"
        }
    },
    "directory": "full-teste",
    "benchmarks": [
        {
            "benchmark": "e3sm",
            "file": "coisa.h5",
            "configuration": {
                "k": "",
                "x": "blob",
                "a": "hdf5",
                "r": "25",
                "o": "ON",
                "netcdf": "../../e3sm/datasets/f_case_866x72_16p.nc"
            }
        },
        {
            "benchmark": "write",
            "file": "test.h5",
            "configuration": {
                "MEM_PATTERN": "CONTIG",
                "FILE_PATTERN": "CONTIG",
                "NUM_PARTICLES": "16 M",
                "TIMESTEPS": "5",
                "DELAYED_CLOSE_TIMESTEPS": "2",
                "COLLECTIVE_DATA": "NO",
                "COLLECTIVE_METADATA": "NO",
                "EMULATED_COMPUTE_TIME_PER_TIMESTEP": "1 s",
                "NUM_DIMS": "1",
                "DIM_1": "16777216",
                "DIM_2": "1",
                "DIM_3": "1",
                "ASYNC_MODE": "NON",
                "CSV_FILE": "output.csv"
            }
        },
        {
```

(continues on next page)

(continued from previous page)

```
            "benchmark": "exerciser",
            "configuration": {
                "numdims": "2",
                "minels": "8 8",
                "nsizes": "3",
                "bufmult": "2 2",
                "dimranks": "8 4"
            }
        },
        {
            "benchmark": "exerciser",
            "configuration": {
                "numdims": "2",
                "minels": "8 8",
                "nsizes": "3",
                "bufmult": "2 2",
                "dimranks": "8 4"
            }
        },
        {
            "benchmark": "exerciser",
            "configuration": {
                "numdims": "2",
                "minels": "8 8",
                "nsizes": "3",
                "bufmult": "2 2",
                "dimranks": "8 4"
            }
        },
        {
            "benchmark": "exerciser",
            "configuration": {
                "numdims": "2",
                "minels": "8 8",
                "nsizes": "3",
                "bufmult": "2 2",
                "dimranks": "8 4"
            }
        },
        {
            "benchmark": "exerciser",
            "configuration": {
                "numdims": "2",
                "minels": "8 8",
                "nsizes": "3",
                "bufmult": "2 2",
                "dimranks": "8 4"
            }
        },
        {
            "benchmark": "read",
            "file": "test.h5",
```

(continues on next page)

(continued from previous page)

```
            "configuration": {
                "MEM_PATTERN": "CONTIG",
                "FILE_PATTERN": "CONTIG",
                "NUM_PARTICLES": "16 M",
                "TIMESTEPS": "5",
                "DELAYED_CLOSE_TIMESTEPS": "2",
                "COLLECTIVE_DATA": "NO",
                "COLLECTIVE_METADATA": "NO",
                "EMULATED_COMPUTE_TIME_PER_TIMESTEP": "1 s",
                "NUM_DIMS": "1",
                "DIM_1": "16777216",
                "DIM_2": "1",
                "DIM_3": "1",
                "ASYNC_MODE": "NON",
                "CSV_FILE": "output.csv"
            }
        },
        {
            "benchmark": "write",
            "file": "test-two.h5",
            "configuration": {
                "MEM_PATTERN": "CONTIG",
                "FILE_PATTERN": "CONTIG",
                "NUM_PARTICLES": "2 M",
                "TIMESTEPS": "20",
                "DELAYED_CLOSE_TIMESTEPS": "2",
                "COLLECTIVE_DATA": "NO",
                "COLLECTIVE_METADATA": "NO",
                "EMULATED_COMPUTE_TIME_PER_TIMESTEP": "1 s",
                "NUM_DIMS": "1",
                "DIM_1": "16777216",
                "DIM_2": "1",
                "DIM_3": "1",
                "ASYNC_MODE": "NON",
                "CSV_FILE": "output.csv"
            }
        },
        {
            "benchmark": "metadata",
            "file": "hdf5_iotest.h5",
            "configuration": {
                "version": "0",
                "steps": "20",
                "arrays": "500",
                "rows": "100",
                "columns": "200",
                "process-rows": "2",
                "process-columns": "2",
                "scaling": "weak",
                "dataset-rank": "4",
                "slowest-dimension": "step",
                "layout": "contiguous",
```

(continues on next page)

(continued from previous page)

```
                "mpi-io": "independent",
                "csv-file": "hdf5_iotest.csv"
            }
        }
    ]
}
```

For a description of all the options available in each benchmark, please refer to their entries in the documentation.

When the `--debug` option is enabled, you can expect an output similar to:

```
2021-10-25 16:31:24,866 h5bench - INFO - Starting h5bench Suite
2021-10-25 16:31:24,889 h5bench - INFO - Lustre support detected
2021-10-25 16:31:24,889 h5bench - DEBUG - Lustre stripping configuration: lfs setstripe -
↪S 1M -c 4 full-teste
2021-10-25 16:31:24,903 h5bench - INFO - h5bench [write] - Starting
2021-10-25 16:31:24,903 h5bench - INFO - h5bench [write] - DIR: full-teste/504fc233/
2021-10-25 16:31:24,904 h5bench - INFO - Parallel setup: srun --cpu_bind=cores -n 4
2021-10-25 16:31:24,908 h5bench - INFO - srun --cpu_bind=cores -n 4 build/h5bench_write␣
↪full-teste/504fc233/h5bench.cfg full-teste/test.h5
2021-10-25 16:31:41,670 h5bench - INFO - SUCCESS
2021-10-25 16:31:41,754 h5bench - INFO - Runtime: 16.8505464 seconds (elapsed time,␣
↪includes allocation wait time)
2021-10-25 16:31:41,755 h5bench - INFO - h5bench [write] - Complete
2021-10-25 16:31:41,755 h5bench - INFO - h5bench [exerciser] - Starting
2021-10-25 16:31:41,755 h5bench - INFO - h5bench [exerciser] - DIR: full-teste/247659d1/
2021-10-25 16:31:41,755 h5bench - INFO - Parallel setup: srun --cpu_bind=cores -n 4
2021-10-25 16:31:41,756 h5bench - INFO - srun --cpu_bind=cores -n 4 build/h5bench_
↪exerciser --numdims 2  --minels 8 8  --nsizes 3  --bufmult 2 2  --dimranks 8 4
2021-10-25 16:31:49,174 h5bench - INFO - SUCCESS
2021-10-25 16:31:49,174 h5bench - INFO - Finishing h5bench Suite
```

Cori

In Cori you need to load Python and its libraries for the main `h5bench` script to work. For manual execution of each benchmark that is not required.

```
module load python
```

In case you are running on Cori and the benchmark fails with an MPI message indicating no support for multiple threads, make sure you define:

```
export MPICH_MAX_THREAD_SAFETY="multiple"
```

2.2 Manual Execution

If you prefer, you can execute each benchmark manually. In this scenario, you will be responsible for generating the input configuration file needed for each benchmark in the suite, ensuring it follows the pre-defined format unique for each one.

If you want to use HDF5 VOL connectors or tune the file system configuration, `h5bench` will *not* take care of that. Remember that not all benchmarks in the suite have support for VOL connectors yet.

CHAPTER

THREE

READ / WRITE

This set of benchmarks contains an I/O kernel developed based on a particle physics simulation's I/O pattern (VPIC-IO for writing data in a HDF5 file) and on a big data clustering algorithm (BDCATS-IO for reading the HDF5 file VPIC-IO wrote).

3.1 Configuration

You can configure the `h5bench_write` and `h5bench_read` benchmarks with the following options. Notice that if you use the `configuration.json` approach to define the runs for `h5bench`, we will automatically generate the final configuration file based on the options you provide in the JSON file. For standalone usage of this benchmark, you can check the input format at the end of this document and refer to its documentation.

Parameter	Description
`MEM_PATTERN`	Options: `CONTIG`, `INTERLEAVED`, and `STRIDED`
`FILE_PATTERN`	Options: `CONTIG` and `STRIDED`
`TIMESTEPS`	The number of iterations
`EMULATED_COMPUTE_TIME_PER_TIMESTEP`	Sleeps after each iteration to emulate computation
`NUM_DIMS`	The number of dimensions, valid values are 1, 2 and 3
`DIM_1`	The dimensionality of the source data
`DIM_2`	The dimensionality of the source data
`DIM_3`	The dimensionality of the source data

For `MEM_PATTERN`, `CONTIG` represents arrays of basic data types (i.e., int, float, double, etc.); `INTERLEAVED` represents an array of structure (AOS) where each array element is a C struct; and `STRIDED` represents a few elements in an array of basic data types that are separated by a constant stride. `STRIDED` is supported only for 1D arrays.

For `FILE_PATTERN`, `CONTIG` represents a HDF5 dataset of basic data types (i.e., int, float, double, etc.); `INTERLEAVED` represents a dataset of a compound datatype;

For `EMULATED_COMPUTE_TIME_PER_TIMESTEP`, you *must* provide the time unit (e.g. `10 s`, `100 ms`, or `5000us`) to ensure correct behavior.

For `DIM_2` and `DIM_3` if **unused**, you should set both as `1`. Notice that the total number of particles will be given by `DIM_1 * DIM_2 * DIM_3`. For example, `DIM_1=1024`, `DIM_2=256`, `DIM_3=1` is a valid setting for a 2D array and it will generate `262144` particles.

A set of sample configuration files can be found in the `samples/` diretory in GitHub.

3.1.1 READ Settings (h5bench_read)

Parameter	Description
READ_OPTION	Options: FULL, PARTIAL, and STRIDED

For the PARTIAL option, the benchmark will read only the first TO_READ_NUM_PARTICLES particles.

3.1.2 Asynchronous Settings

Parameter	Description
MODE	Options: SYNC or ASYNC
IO_MEM_LIMIT	Memory threshold to determine when to execute I/O
DELAYED_CLOSE_TIMESTEPS	Groups and datasets will be closed later.

The IO_MEM_LIMIT parameter is optional. Its default value is 0 and it requires ASYNC, i.e., it only works in asynchronous mode. This is the memory threshold used to determine when to actually execute the I/O operations. The actual I/O operations (data read/write) will not be executed until the timesteps associated memory reachs the threshold, or the application run to the end.

For the ASYNC mode to work you **must** define the necessay HDF5 ASYNC-VOL connector. For more information about it, refer to its documentation.

3.1.3 Compression Settings

Parameter	Description
COMPRESS	*YES* or *NO* (optional) enables parralel compression
CHUNK_DIM_1	Chunk dimension
CHUNK_DIM_2	Chunk dimension
CHUNK_DIM_3	Chunk dimension

Compression is only applicable for h5bench_write. It has not effect for h5bench_read. When enabled the chunk dimensions parameters (CHUNK_DIM_1, CHUNK_DIM_2, CHUNK_DIM_3) are required. The chunk dimension settings should be compatible with the data dimensions, i.e., they must have the same rank of dimensions, and chunk dimension size cannot be greater than data dimension size. Extra chunk dimensions have no effect and should be set to 1.

Warning: There is a known bug on HDF5 parallel compression that could cause the system run out of memory when the chunk amount is large (large number of particle and very small chunk sizes). On Cori Hasswell nodes, the setting of 16M particles per rank, 8 nodes (total 256 ranks), 64 * 64 chunk size will crash the system by runing out of memory, on single nodes the minimal chunk size is 4 * 4.

3.1.4 Collective Operation Settings

Parameter	Description
COLLECTIVE_DATA	Enables collective operation (default is NO)
COLLECTIVE_METADATA	Enables collective HDF5 metadata (default is NO)

Both COLLECTIVE_DATA and COLLECTIVE_METADATA parameters are optional.

3.1.5 CSV Settings

Performance results will be written to this file and standard output once a file name is provided.

Parameter	Description
CSV_FILE	CSV file name to store benchmark results

3.2 Supported Patterns

Attention: Not every pattern combination is covered by the benchmark. Supported benchmark parameter settings are listed below.

3.2.1 Supported Write Patterns (h5bench_write)

The I/O patterns include array of structures (AOS) and structure of arrays (SOA) in memory as well as in file. The array dimensions are 1D, 2D, and 3D for the write benchmark. This defines the write access pattern, including CONTIG (contiguous), INTERLEAVED and STRIDED for the source (the data layout in the memory) and the destination (the data layout in the resulting file). For example, MEM_PATTERN=CONTIG and FILE_PATTERN=INTERLEAVED is a write pattern where the in-memory data layout is contiguous (see the implementation of prepare_data_contig_2D() for details) and file data layout is interleaved by due to its compound data structure (see the implementation of data_write_contig_to_interleaved() for details).

- 4 patterns for both 1D and 2D array write (NUM_DIMS=1 or NUM_DIMS=2)

```
'MEM_PATTERN': 'CONTIG'
'FILE_PATTERN': 'CONTIG'
```

```
'MEM_PATTERN': 'CONTIG'
'FILE_PATTERN': 'INTERLEAVED'
```

```
'MEM_PATTERN': 'INTERLEAVED'
'FILE_PATTERN': 'CONTIG'
```

```
'MEM_PATTERN': 'INTERLEAVED'
'FILE_PATTERN': 'INTERLEAVED'
```

- 1 pattern for 3D array (NUM_DIMS=3)

```
'MEM_PATTERN': 'CONTIG'
'FILE_PATTERN': 'CONTIG'
```

- 1 strided pattern for 1D array (NUM_DIMS=1)

```
'MEM_PATTERN': 'CONTIG'
'FILE_PATTERN': 'STRIDED'
```

3.2.2 Supported Read Patterns (h5bench_read)

- 1 pattern for 1D, 2D and 3D read (NUM_DIMS=1 or NUM_DIMS=2)

Contiguously read through the whole data file:

```
'MEM_PATTERN': 'CONTIG'
'FILE_PATTERN': 'CONTIG'
'READ_OPTION': 'FULL'
```

- 2 patterns for 1D read

Contiguously read the first TO_READ_NUM_PARTICLES elements:

```
'MEM_PATTERN': 'CONTIG'
'FILE_PATTERN': 'CONTIG'
'READ_OPTION': 'PARTIAL'
```

```
'MEM_PATTERN': 'CONTIG'
'FILE_PATTERN': 'STRIDED'
'READ_OPTION': 'STRIDED'
```

3.3 Understanding the Output

The metadata and raw data operations are timed separately, and the overserved time and I/O rate are based on the total time.

Sample output of h5bench_write:

```
==================  Performance results  =================
Total emulated compute time 4000 ms
Total write size = 2560 MB
Data preparation time = 739 ms
Raw write time = 1.012 sec
Metadata time = 284.990 ms
H5Fcreate() takes 4.009 ms
H5Fflush() takes 14.575 ms
H5Fclose() takes 4.290 ms
Observed completion time = 6.138 sec
Raw write rate = 2528.860 MB/sec
Observed write rate = 1197.592 MB/sec
```

Sample output of h5bench_read:

```
=================  Performance results  =================
Total emulated compute time = 4 sec
Total read size = 2560 MB
```

(continues on next page)

(continued from previous page)

```
Metadata time = 17.523 ms
Raw read time = 1.201 sec
Observed read completion time = 5.088 sec
Raw read rate = 2132.200 MB/sec
Observed read rate = 2353.605225 MB/sec
```

3.4 Known Issues

Warning: In Cori/NERSC or similar platforms that use Cray-MPICH library, if you encouter a failed assertion regarding support for `MPI_THREAD_MULTIPLE` you should define the following environment variable:

```
export MPICH_MAX_THREAD_SAFETY="multiple"
```

Warning: If you're trying to run the benchmark with the HDF5 VOL ASYNC connector in MacOS and are getting segmentation fault (from `ABT_thread_create`), please try to set the following environment variable:

```
export ABT_THREAD_STACKSIZE=100000
```

CHAPTER

FOUR

METADATA STRESS

The Metadata Stress benchmark (`h5bench_hdf5_iotest`) is a simple I/O performance tester for HDF5. Its purpose is to assess the performance variability of a set of logically equivalent HDF5 representations of a common pattern. The test repeatedly writes (and reads) in parallel a set of 2D array variables in a tiled fashion, over a set of time steps. For more information referer to HDF Group GitHub repository. We modified this benchmark slightly so to be able to specify the config file location, everything else remains untouched.

4.1 Configuration

You can configure the Metadata Stress test with the following options. Notice that if you use the `configuration.json` approach to define the runs for `h5bench`, we will automatically generate the final configuration file based on the options you provide in the JSON file. For standalone usage of this benchmark, you can check the input format at the end of this document and refer to its documentation.

Parameter	Description
`steps`	Number of steps
`arrays`	Number of arrays
`rows`	Total number of array rows for strong scaling. Number of array rows per block for weak scaling.
`columns`	Total number of array columns for strong scaling. Number of array columns per block for weak scaling.
`process-rows`	Number of MPI-process rows: rows % proc-rows == 0 for strong scaling
`process-columns`	Number of MPI-process columns: columns % proc-columns == 0 for strong scaling
`scaling`	Scaling ([weak, strong])
`dataset-rank`	Rank of the dataset(s) in the file ([2, 3, 4])
`slowest-dimension`	Slowest changing dimension ([step, array])
`layout`	HDF5 dataset layout ([contiguous, chunked]
`mpi-io`	MPI I/O mode ([independent, collective])
`hdf5-file`	HDF5 output file name
`csv-file`	CSV results file name

4.1.1 JSON Configuration (recomended)

To run an instance of Metadata Stress Test benchmark you need to include the following in the `benchmarks` property of your `configuration.json` file:

```
{
    "benchmark": "metadata",
    "file": "hdf5_iotest.h5",
    "configuration": {
        "version": "0",
        "steps": "20",
        "arrays": "500",
        "rows": "100",
        "columns": "200",
        "process-rows": "2",
        "process-columns": "2",
        "scaling": "weak",
        "dataset-rank": "4",
        "slowest-dimension": "step",
        "layout": "contiguous",
        "mpi-io": "independent",
        "csv-file": "hdf5_iotest.csv"
    }
}
```

4.1.2 Standalone Configuration

For standalone usage of this benchmark, this is the observed input configuration you should provide to the `h5bench_hdf5_iotest` executable.

```
[DEFAULT]
version = 0
steps = 20
arrays = 500
rows = 100
columns = 200
process-rows = 1
process-columns = 1
scaling = weak
dataset-rank = 4
slowest-dimension = step
layout = contiguous
mpi-io = independent
hdf5-file = hdf5_iotest.h5
csv-file = hdf5_iotest.csv
```

CHAPTER

FIVE

AMREX

AMReX is a software framework for massively parallel, block-structured adaptive mesh refinement (AMR) applications.

You can find more information in AMReX-Codes GitHub repository.

5.1 Configuration

You can configure the AMReX HDF5 benchmark with the following options. Notice that if you use the `configuration.json` approach to define the runs for `h5bench`, we will automatically generate the final configuration file based on the options you provide in the JSON file. For standalone usage of this benchmark, you can check the input format at the end of this document and refer to its documentation.

Parameter	Description
`ncells`	Domain size
`max_grid_size`	The maximum allowable size of each subdomain (used for parallel decomposal)
`nlevs`	Number of levels
`ncomp`	Number of components in the multifabs
`nppc`	Number of particles per cell
`nplotfile`	Number of plot files to write
`nparticlefile`	Number of particle files to write
`sleeptime`	Time to sleep before each write
`restart_check`	Whether to check the correctness of checkpoint/restart
`grids_from_file`	Enable AMReX to read grids from file
`ref_ratio_file`	Refinement ratios for different AMReX refinement levels
`hdf5compression`	Define the HDF5 compression algorithm to use

5.1.1 JSON Configuration (recomended)

To run an instance of AMReX HDF5 benchmark you need to include the following in the `benchmarks` property of your `configuration.json` file:

```
{
    "benchmark": "amrex",
    "file": "amrex.h5",
    "configuration": {
        "ncells": "64",
        "max_grid_size": "8",
```

(continues on next page)

(continued from previous page)

```
        "nlevs": "1",
        "ncomp": "6",
        "nppc": "2",
        "nplotfile": "2",
        "nparticlefile": "2",
        "sleeptime": "2",
        "restart_check": "1",
        "hdf5compression": "ZFP_ACCURACY#0.001"
    }
}
```

To read grids from file you need to set: `grids_from_file`, `nlevels`, and `ref_ratio_file`.

```
{
    "benchmark": "amrex",
    "file": "amrex.h5",
    "configuration": {
        "ncells": "64",
        "max_grid_size": "8",
        "nlevs": "1",
        "ncomp": "6",
        "nppc": "2",
        "nplotfile": "2",
        "nparticlefile": "2",
        "sleeptime": "2",
        "restart_check": "1",
        "hdf5compression": "ZFP_ACCURACY#0.001",
        "nlevs": "3",
        "grids_from_file": "1",
        "ref_ratio_file": "4 2"
    }
}
```

5.1.2 HDF5 ASYNC VOL Connector

AMReX supports the HDF5 ASYNC VOL connector. To enable it, you should specify in the `vol` property of you `configuration.json` file: the required library paths, the VOL ASYNC source path, and the connector setup.

```
"vol": {
    "library": "/vol-async/src:/hdf5-async-vol-register-install/lib:/argobots/install/
↪lib:/hdf5-install/install:",
    "path": "/vol-async/src",
    "connector": "async under_vol=0;under_info={}"
}
```

5.1.3 Standalone Configuration

For standalone usage of this benchmark, this is the observed input configuration you should provide to the `h5bench_amrex` executable.

```
ncells = 64
max_grid_size = 8
nlevs = 1
ncomp = 6
nppc = 2
nplotfile = 2
nparticlefile = 2
sleeptime = 2
restart_check = 1

# Uncomment to read grids from file
# nlevs = 3
# grids_from_file = 1
# ref_ratio_file = 4 2

# Uncomment to enable compression
# hdf5compression=ZFP_ACCURACY#0.001

directory = .
```

CHAPTER

SIX

OPENPMD

OpenPMD is an open meta-data schema that provides meaning and self-description for data sets in science and engineering.

The openPMD-api library provides a reference API for openPMD data handling. In the h5bench Benchmarking Suite we provide support for the write and read parallel benchmarks with HDF5 backend. You can find more information in OpenPMD documentation.

6.1 Configuration

You can configure the openPMD write HDF5 benchmark with the following options. Notice that if you use the `configuration.json` approach to define the runs for `h5bench`, we will automatically generate the final configuration file based on the options you provide in the JSON file. For standalone usage of this benchmark, you can check the input format at the end of this document and refer to its documentation.

Parameter	Description
`operation`	Operation: `write` or `read`
`fileLocation`	Directory where the file will be written to or read from

When running with the `write` operation, you have to define the following options:

`dim` Number of dimensions (`1`, `2`, or `3`) `balanced` Should it use a balanced load? (`true` or `false`) `ratio` Particle to mesh ratio `steps` Number of iteration steps `minBlock` Meshes are viewed as grid of mini blocks `grid` Grid based on the mini block

When running with the `read` operation, you have to define the pattern:

`pattern` Read access pattern

The `minBlock` and `grid` parameters must include the values for each of the `dim` dimensions. For example, if `"dim": "3"` (for a 3D mesh) `minBlock` should contain three values, one for each dimenseion `"16 32 32"` and `grid` (which is based on the mini block) should also contain three values, one for each dimension `"32 32 16"`.

For the `pattern` attribute for read you can chose:

- `m`: metadata onlune
- `sx`: slice of the 'rho' mesh in the x-axis (eg. `x=0`)
- `sy`: slice of the 'rho' mesh in the y-axis (eg. `y=0`)
- `sz`: slice of the 'rho' mesh in the z-axis (eg. `z=0`)
- `fx`: slice of the 3D magnetic field in the x-axis (eg. `x=0`)
- `fy`: slice of the 3D magnetic field in the y-axis (eg. `y=0`)

- `fz`: slice of the 3D magnetic field in the z-axis (eg. `z=0`)

6.1.1 JSON Configuration (recomended)

To run an instance of openPMD HDF5 benchmark you need to include the following in the `benchmarks` property of your `configuration.json` file:

```
{
    "benchmark": "openpmd",
    "configuration": {
        "operation": "write",
        "dim": "3",
        "balanced": "true",
        "ratio": "1",
        "steps": "1",
        "minBlock": "8 16 16",
        "grid": "16 16 8"
    }
},
{
    "benchmark": "openpmd",
    "configuration": {
        "operation": "read",
        "dim": "3",
        "balanced": "true",
        "ratio": "1",
        "steps": "1",
        "minBlock": "8 16 16",
        "grid": "16 16 8"
    }
}
```

6.1.2 Standalone Configuration

For standalone usage of this benchmark, this is the observed input configuration you should provide to the `h5bench_openpmd_write`.

```
dim=3
balanced=true
ratio=1
steps=10
minBlock=16 32 32
grid=32 32 16
```

For the `h5bench_openpmd_read`, you need to provide two arguments: the file prefix and the pattern.

CHAPTER

SEVEN

EXERCISER

Attention: For more-detailed instructions of how to build and run the exerciser code on specific machines (at ALCF), see the Exerciser/BGQ/VESTA_XL/README.md and Exerciser/BGQ/THETA/README.md directories of this repository. Those README files also include instructions for building the CCIO and develop versions of HDF5 for use with this benchmark.

The **HDF5 Exerciser Benchmark** creates an HDF5 use case with some ideas/code borrowed from other benchmarks (namely IOR, VPICIO and FLASHIO). Currently, the algorithm does the following in parallel over all MPI ranks:

- For each rank, a local data buffer (with dimensions given by `numdims` is initialized with minNEls double-precision elements in each dimension
- If the `derivedtype` flag is used, a second local dataset is also specified with a derived data type a-signed to each element
- For a given number of iterations (hardcoded as `NUM_ITERATIONS`):
  - Open a file, create a top group, set the MPI-IO transfer property, and (optionally) add a simple attribute string to the top group
  - Create memory and file dataspaces with hyperslab selections for simple rank-ordered offsets into the file. The `rshift` option can be used to specify the number of rank positions to shift the write position in the file (the read will be shifted twice this amount to avoid client-side caching effects
  - Write the data and close the file
  - Open the file, read in the data, and check correctness (if dataset is small enough)
  - Close the dataset (but not the file)
  - If the second (derived-type) data set is specified: (1) create a derived type, (2) open a new data set with the same number of elements and dimension, (3) write the data and (4) close everything
- Each dimension of curNEls is then multiplied by each dimension of bufMult, and the previous steps (the loop over `NUM_ITERATIONS`) are repeated. This outer loop over local buffer sizes is repeated a total of nsizes times

7.1 Configuration

You can configure the `h5bench_write` and `h5bench_read` benchmarks with the following options. Notice that if you use the `configuration.json` approach to define the runs for `h5bench`, we will automatically generate the final configuration file based on the options you provide in the JSON file. For standalone usage of this benchmark, you can check the input format at the end of this document and refer to its documentation.

7.1.1 Required

Parameter	Description
`numdims <x>`	Dimension of the datasets to write to the HDF5 file
`minels <x> ... <x>`	Min number of double elements to write in each dim of the dataset (one value for each dimension)

7.1.2 Optional

Parameter	Description
`nsizes <x>`	How many buffer sizes to use (Code will start with minbuf and loop through nsizes iterations, with the buffer size multiplied by bufmult in each dim, for each iteration)
`bufmult <x> ... <x>`	Constant, for each dimension, used to multiply the buffer [default: 2 2 …]
`metacoll`	Whether to set meta data collective usage [default: False]
`derivedtype`	Whether to create a second data set containing a derived type [default: False]
`addattr`	Whether to add attributes to group 1 [default: False]
`indepio`	Whether to use independant I/O (not MPI-IO) [default: False]
`keepfile`	Whether to keep the file around after the program ends for futher analysis, otherwise deletes it [default: False]
`usechunked`	Whether to chunk the data when reading/writing [default: False]
`maxcheck <x>`	Maximum buffer size (in bytes) to validate. Note that all buffers will be vaidated if this option is not set by this command-line argument [default: Inf]
`memblock <x>`	Define the block size to use in the local memory buffer (local buffer is always 1D for now, Note: This currently applies to the 'double' dataset only) [default: local buffer size]
`memstride <x>`	Define the stride of the local memory buffer (local buffer is always 1D for now, Note: This currently applies to the 'double' dataset only) [default: local buffer size]
`fileblocks <x> ...<x>`	Block sizes to use in the file for each dataset dimension (Note: This currently applies to the 'double' dataset only) [default: 1 … 1]
`filestrides <x> ...<x>`	Stride dist. to use in the file for each dataset dimension (Note: This currently applies to the 'double' dataset only) [default: 1 … 1]

The exerciser also allows the MPI decomposition to be explicitly defined:

Parameter	Description
`dimranks <x> ...<x>`	MPI-rank division in each dimension. Note that, if not set, decomposition will be in 1st dimension only.

7.2 Exerciser Basics

In the simplest case, the Exerciser code will simply write and then read an n-dimensional double-precision dataset in parallel (with all the necessary HDF5 steps in between). At a minimum, the user must specify the number of dimensions to use for this dataset (using the `numdims` flag), and the size of each dimension (using the `minels` flag). By default, the maximum number of dimensions allowed by the code is set by `MAX_DIM` (currently 4, but can be modified easily). Note that the user is specifying the number of elements to use in each dimension with `minels`. Therefore, the local buffer size is the product of the dimension sizes and `sizeof(double)` (and the global dataset in the file is a product of the total MPI ranks and the local buffer size). As illustrated in Fig. 1, the mapping of ranks to hyper-slabs in the global dataset can be specified with the `dimranks` flag (here, Example 1 is the default decomposition, while Example 2 corresponds to: `"dimranks": "2 2"`). This flag simply allows the user to list the number of spatial decompositions in each dimension of the global dataset, and requires that the product of the input to be equal to the total number of MPI ranks.

Fig. 1: Fig. 1 - Illustration of different local-to-global dataset mapping options.

Note: Authors:

- Richard J. Zamora (rzamora@anl.gov)
- Paul Coffman (pcoffman@anl.gov)
- Venkatram Vishwanath (venkat@anl.gov)

CHAPTER

EIGHT

E3SM

E3SM-IO is the parallel I/O kernel from the E3SM climate simulation model. It makes use of PIO library which is built on top of PnetCDF.

This benchmark currently has two cases from E3SM, namely F and G cases. The F case uses three unique data decomposition patterns shared by 388 2D and 3D variables (2 sharing Decomposition 1, 323 sharing Decomposition 2, and 63 sharing Decomposition 3). The G case uses 6 data decompositions shared by 52 variables (6 sharing Decomposition 1, 2 sharing Decomposition 2, 25 sharing Decomposition 3, 2 sharing Decomposition 4, 2 sharing Decomposition 5, and 4 sharing Decomposition 6).

You can find more information in Parallel-NetCDF GitHub repository.

8.1 Configuration

You can configure the ES3M-IO benchmark with the following options. Notice that if you use the `configuration.json` approach to define the runs for `h5bench`, we will automatically generate the final configuration file based on the options you provide in the JSON file. For standalone usage of this benchmark, you can refer to E3SM-IO repository.

Parameter	Description
`k`	Keep the output files when program exits
`x`	I/O strategy to write (`canonical`, `log`, and `blob`)
`a`	I/O library name to perform write operation (`hdf5`, `hdf5_log`, `hdf5_md`)
`r`	Number of records/time steps for F case h1 file
`o`	Enable write performance evaluation
`netcdf`	Define the HDF5 compression algorithm to use

Warning: h5bench temporarily only supports `-x blob` and `-a hdf5`. If you set other options, they will be overwritten to the supported version.

8.1.1 JSON Configuration (recomended)

To run an instance of AMReX HDF5 benchmark you need to include the following in the `benchmarks` property of your `configuration.json` file:

```
{
    "benchmark": "e3sm",
    "file": "coisa.h5",
    "configuration": {
        "k": "",
        "x": "blob",
        "a": "hdf5",
        "r": "25",
        "o": "ON",
        "netcdf": "../../e3sm/datasets/f_case_866x72_16p.nc"
    }
}
```

CHAPTER

NINE

GPU-IO

These benchmarks extend the *Read / Write benchmarks* with memory transfers to and from GPU memory. Refer to the *Build Instructions* for enabling these benchmarks with CUDA support.

9.1 Configuration

You can configure the `h5bench_cuda_write` and `h5bench_cuda_read` benchmarks with the following options. Notice that if you use the `samples/sync-cuda-write-1d-contig-contig.json` approach to define the runs for `h5bench`, we will automatically generate the final configuration file based on the options you provide in the JSON file. For standalone usage of this benchmark, you can check the input format at the end of this document and refer to its documentation.

Parameter	Description
MEM_PATTERN	Options: CONTIG, INTERLEAVED, and STRIDED
FILE_PATTERN	Options: CONTIG and STRIDED
TIMESTEPS	The number of iterations
EMULATED_COMPUTE_TIME_PER_TIMESTEP	Sleeps after each iteration to emulate computation
NUM_DIMS	The number of dimensions, valid values are 1, 2 and 3
DIM_1	The dimensionality of the source data
DIM_2	The dimensionality of the source data
DIM_3	The dimensionality of the source data
DIM_3	The dimensionality of the source data
FILE_PER_PROC	YES to enable file per process mode (NO is not supported with GPU-IO)

For `MEM_PATTERN`, `CONTIG` represents arrays of basic data types (i.e., int, float, double, etc.); `INTERLEAVED` represents an array of structure (AOS) where each array element is a C struct; and `STRIDED` represents a few elements in an array of basic data types that are separated by a constant stride. `STRIDED` is supported only for 1D arrays.

For `FILE_PATTERN`, `CONTIG` represents a HDF5 dataset of basic data types (i.e., int, float, double, etc.); `INTERLEAVED` represents a dataset of a compound datatype;

For `EMULATED_COMPUTE_TIME_PER_TIMESTEP`, you *must* provide the time unit (e.g. `10 s`, `100 ms`, or `5000us`) to ensure correct behavior.

For `DIM_2` and `DIM_3` if **unused**, you should set both as `1`. Notice that the total number of particles will be given by `DIM_1 * DIM_2 * DIM_3`. For example, `DIM_1=1024`, `DIM_2=256`, `DIM_3=1` is a valid setting for a 2D array and it will generate `262144` particles.

A set of sample configuration files can be found in the `samples/` diretory in GitHub.

9.1.1 READ Settings (h5bench_cuda_read)

Parameter	Description
READ_OPTION	Options: FULL, PARTIAL, and STRIDED

For the PARTIAL option, the benchmark will read only the first TO_READ_NUM_PARTICLES particles.

9.1.2 GPUDirect Storage with GDS VFD

With the HDF5 GDS VFD (https://github.com/hpc-io/vfd-gds), you can benchmark GPUDirect Storage. Note that not all NVIDIA GPUs and filesystems support GDS (see https://docs.nvidia.com/gpudirect-storage for more details on supported platforms). Currently, this mode is only supported by running the benchmark with a config file with the follow parameter set, manually.

Parameter	Description
DYNAMIC_VFD_NAME	Options: gds

9.1.3 CSV Settings

Performance results will be written to this file and standard output once a file name is provided.

Parameter	Description
CSV_FILE	CSV file name to store benchmark results

9.2 Understanding the Output

The metadata and raw data operations are timed separately, and the overserved time and I/O rate are based on the total time.

Sample output of h5bench_cuda_write:

```
=================== Performance Results ==================
Total number of ranks: 1
Total emulated compute time: 4.000 s
Total write size: 2.500 GB
Raw h2d time = 82.600 s
Raw d2h time = 46.584 s
Raw write time: 2.003 s
Raw Full write time (inc. d2h) = 246.879 s
Metadata time: 0.002 s
H5Fcreate() time: 0.002 s
H5Fflush() time: 0.000 s
H5Fclose() time: 0.000 s
Observed completion time: 7.311 s
SYNC Raw h2d rate: 30.992 MB/s
SYNC Raw d2h rate: 54.954 MB/s
SYNC Raw write rate: 1.248 GB/s
SYNC Raw Full write rate (inc. d2h): 10.369 MB/s
```

(continues on next page)

(continued from previous page)

```
SYNC Observed write rate: 773.215 MB/s
============================================================
```

Sample output of `h5bench_cuda_read`:

```
=================== Performance Results ==================
Total number of ranks: 1
Total emulated compute time: 4.000 s
Total read size: 2.500 GB
Raw h2d time = 0.697 s
Raw d2h time = 0.494 s
Raw read time: 1.155 s
Raw Full read time (inc. h2d) = 1.851 s
Metadata time: 0.002 s
Observed read completion time: 6.349 s
SYNC Raw h2d rate: 3.587 GB/s
SYNC Raw d2h rate: 5.058 GB/s
SYNC Raw read rate: 2.165 GB/s
SYNC Raw Full read rate (inc. d2h) = 1.350 GB/s
SYNC Observed read rate: 1.063 GB/s
============================================================
```

CHAPTER

TEN

WAYS TO CONTRIBUTE

We appreciate your interest in **h5bench**, and thank you for taking the time to contribute!

We have compiled a set of instructions to help us make h5bench even better.

10.1 Reporting bugs

You can open a new issue using our GitHub issue tracker. If you run into an issue, please search first to ensure the issue has not been reported before. Open a new issue only if you have not found anything similar to your issue. Please, try to provide as much information as possible to reproduce your bug quickly.

10.2 Suggesting enhancements

You can use our GitHub issue tracker to describe your proposed feature. Please, provide the necessary context, covering why it is needed and what problem does it solve.

10.3 Adding new benchmarks

We provide a set of instructions on how to add new benchmarks to the h5bench Benchmarking Suite. However, please notice that you might require some changes depending on how your benchmark work. You can contribute in two ways:

- **Adding existing benchmarks as submodules**: We plan to support only the version included in the original PR, based on its commit hash. Updates on the submodule require the contributor's help to ensure we can smoothly upgrade the available version without breaking existing features (both in the benchmark and in h5bench).
- **Adding newly developed benchmarks**: The community may perform maintenance, requiring you to provide comprehensive documentation (in code and usage) and examples to understand the benchmark module.

10.3.1 Example

To illustrate how you can add a new benchmark using the submodule aprroach we will use AMReX:

1. You need to include the AMReX repository as a submodule:

```
git submodule add https://github.com/AMReX-Codes/amrex amrex
```

2. For this benchmark, we need some libraries to be compiled and available as well, so we will need to modify our `CMakeLists.txt`, so it builds that subdirectory:

```
set(AMReX_HDF5 YES)
set(AMReX_PARTICLES YES)
set(AMReX_MPI_THREAD_MULTIPLE YES)
add_subdirectory(amrex)
```

3. AMReX comes with several other benchmarks. Still, since we are only interested in the HDF5 one, we will only compile that code. For that, we will need to add the following to our `CMakeLists.txt`. This is based on how that benchmark is normally compiled within AMReX.

```
set(amrex_src amrex/Tests/HDF5Benchmark/main.cpp)
add_executable(h5bench_amrex ${amrex_src})
```

4. Be sure to follow the convention of naming the executable as `h5bench_` plus the benchmark name, e.g. `h5bench_amrex`.

5. If you are going to provide support for the HDF5 async VOL connector with explicit implementation (which require changes in the original code), make sure you link the required libraries (`asynchdf5` and `h5async`):

```
if(WITH_ASYNC_VOL)
        set(AMREX_USE_HDF5_ASYNC YES)
        target_link_libraries(h5bench_amrex hdf5 z m amrex asynchdf5 h5async MPI::MPI_C)
else()
        target_link_libraries(h5bench_amrex hdf5 z m amrex MPI::MPI_C)
endif()
```

6. The last step is to update the `h5bench` Python-based script to handle the new benchmark. On the top of the file, add the path of your benchmark:

```
H5BENCH_AMREX = 'h5bench_amrex'
```

Update the `run()` function that iterates over the `benchmarks` property list defined by the user in the `configuration.json` file to accept the new benchmark name:

```
elif name == 'amrex':
    self.run_amrex(id, benchmark[name], setup['vol'])
```

You then need to define the `run_` function for the benchmark you're adding. The most important part is translating the configuration defined in the `configuration.json` file into a format accepted by your benchmark (e.g., a file, a JSON, command line). For AMReX, it requires an `amrex.ini` file with key-value configurations defined in the format `key = value`, one per line:

```
# Create the configuration file for this benchmark
with open(configuration_file, 'w+') as f:
    for key in configuration:
        f.write('{} = {}\n'.format(key, configuration[key]))

    f.write('directory = {}\n'.format(file))
```

If you plan to support the HDF5 async VOL connector, make sure you can `enable_vol()` and `disable_vol()` at the beginning and end of this `run_` function.

Here you can check an example of the complete `run_amrex` function:

```
def run_amrex(self, id, setup, vol):
    """Run the AMReX benchmark."""
```

(continues on next page)

(continued from previous page)

```
    self.enable_vol(vol)

    try:
        start = time.time()

        file = '{}/{}'.format(self.directory, setup['file'])
        configuration = setup['configuration']

        configuration_file = '{}/{}/amrex.ini'.format(self.directory, id)

        # Create the configuration file for this benchmark
        with open(configuration_file, 'w+') as f:
            for key in configuration:
                f.write('{} = {}\n'.format(key, configuration[key]))

            f.write('directory = {}\n'.format(file))

        command = '{} {} {}'.format(
            self.mpi,
            self.H5BENCH_AMREX,
            configuration_file
        )

        self.logger.info(command)

        # Make sure the command line is in the correct format
        arguments = shlex.split(command)

        stdout_file_name = 'stdout'
        stderr_file_name = 'stderr'

        with open(stdout_file_name, mode='w') as stdout_file, open(stderr_file_name,␣
↪mode='w') as stderr_file:
            s = subprocess.Popen(arguments, stdout=stdout_file, stderr=stderr_file,␣
↪env=self.vol_environment)
            sOutput, sError = s.communicate()

            if s.returncode == 0:
                self.logger.info('SUCCESS')
            else:
                self.logger.error('Return: %s (check %s for detailed log)', s.returncode,
↪ stderr_file_name)

                if self.abort:
                    self.logger.critical('h5bench execution aborted upon first error')

                    exit(-1)

        end = time.time()

        self.logger.info('Runtime: {:.7f} seconds (elapsed time, includes allocation␣
↪wait time)'.format(end - start))
```

(continues on next page)

(continued from previous page)

```
    except Exception as e:
        self.logger.error('Unable to run the benchmark: %s', e)

    self.disable_vol(vol)
```

7. Make sure you provide some sample JSON configuration files in the `configurations` directory.

Please, feel free to reach us if you have questions!

10.4 Testing

h5bench constantly receives updates and improvements. If you can run the latest version, please consider helping us by reporting your findings, including bugs and performance regressions. Running the benchmarks contained in the h5bench Benchmarking Suite with different configurations and platforms helps us a lot in making it more robust by quickly identifying and solving issues.

CHAPTER

ELEVEN

COPYRIGHT

CHAPTER

TWELVE

LICENSE

(2) Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

(3) Neither the name of the University of California, Lawrence Berkeley National Laboratory, U.S. Dept. of Energy nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

You are under no obligation whatsoever to provide any bug fixes, patches, or upgrades to the features, functionality or performance of the source code ("Enhancements") to anyone; however, if you choose to make your Enhancements available either publicly, or directly to Lawrence Berkeley National Laboratory, without imposing a separate written license agreement for such Enhancements, then you hereby grant the following license: a non-exclusive, royalty-free perpetual license to install, use, modify, prepare derivative works, incorporate into other computer software, distribute, and sublicense such enhancements or derivative works thereof, in binary and source code form.