H5bench

Release 0.1

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

Apr 07, 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.

CHAPTER

ONE

BUILD INSTRUCTIONS

1.1 Build with CMake (recommended)

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:

```
HDF5_HOME: the location you installed HDF5. It should point to a path that look like /
↪path_to_my_hdf5_build/hdf5 and contains include/, lib/ and bin/ subdirectories.
```

1.1.2 Compile with CMake

Assume that the repo is cloned and now you are in the source directory h5bench, run the following simple steps:

```
mkdir build
cd build
cmake ..
make
```

1.1.3 Build to run in async

To run h5bench_vpicio or h5bench_bdcatsio in async mode, you need the develop branchs of BOTH HDF5 and Async-VOL and build H5bench separately.

```
mkdir build
cd build
cmake .. -DWITH_ASYNC_VOL:BOOL=ON -DCMAKE_C_FLAGS="-I/$YOUR_ASYNC_VOL/src -L/$YOUR_ASYNC_
↪VOL/src"
make
```

Necessary environment variable setting:

```
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="$ASYNC_HOME"
export DYLD_LIBRARY_PATH="$HDF5_HOME/lib:$ASYNC_HOME"
```

And all the binaries will be built to the build/directory.

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] [--debug] 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
  --debug     Enable debug mode
```

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.

2.1.1 Configuration

The JSON configuration file has five main properties: *mpi*, *vol*, *file-system*, *directory*, *benchmarks*.

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 in this property, their order, and configuration. You can choose between: *write*, *read*, *metadata*, and *exerciser*.

For the *write* pattern of *h5bench*, you should provide the *file* and the *configuration*:

```
{
   "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"
      }
   }
}
```

For the *read* pattern of *h5bench*, you should provide the *file* and the *configuration*. 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.

```
{
   "read": {
      "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"
      }
   }
}
```

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

```
{
   "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.

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

You can refer to this sample of a complete *configuration.json* file that defined the workflow of the execution of multiple benchmarks from h5bench Suite:

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,␣
```

(continues on next page)

(continued from previous page)

```
↪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 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

BENCHMARK SUITE USAGE

3.1 h5bench_patterns benchmark

Major refactoring is in progress, this document may be out of date. Both h5bench_write and h5bench_read take config and data file path as command line arguments.

```
./h5bench_write my_config.cfg my_data.h5
./h5bench_read my_config.cfg my_data.h5
```

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).

3.1.1 Settings in the Configuration File

The h5bench_write and h5bench_read take parameters in a plain text config file. The content format is **strict**. Unsupported formats :

- Blank/empty lines, including ending lines.
- **Comment symbol(#) follows value immediately:**
  - TIMESTEPS=5# Not supported
  - TIMESTEPS=5 #This is supported
  - TIMESTEPS=5 # This is supported
- **Blank space in assignment**
  - TIMESTEPS=5 # This is supported
  - TIMESTEPS = 5 # Not supported
  - TIMESTEPS =5 # Not supported
  - TIMESTEPS= 5 # Not supported

A template of config file can be found basic_io/sample_config/template.cfg:

```
#=============================================================
#   General settings
NUM_PARTICLES=16 M #16 K/G
TIMESTEPS=5
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1 s #1 ms, 1 min
```

(continues on next page)

(continued from previous page)

```
#============================================================
#   Benchmark data dimensionality
NUM_DIMS=1
DIM_1=16777216
DIM_2=1
DIM_3=1
#============================================================
#   IO pattern settings
IO_OPERATION=READ
#IO_OPERATION=WRITE
MEM_PATTERN=CONTIG
# INTERLEAVED STRIDED
FILE_PATTERN=CONTIG # STRIDED
#============================================================
#   Options for IO_OPERATION=READ
READ_OPTION=FULL # PARTIAL STRIDED
TO_READ_NUM_PARTICLES=4 M
#============================================================
#   Strided access parameters, required for strided access
#STRIDE_SIZE=
#BLOCK_SIZE=
#BLOCK_CNT=
#============================================================
# Collective data/metadata settings
#COLLECTIVE_DATA=NO #Optional, default for NO.
#COLLECTIVE_METADATA=NO #Optional, default for NO.
#============================================================
#    Compression, optional, default is NO.
#COMPRESS=NO
#CHUNK_DIM_1=1
#CHUNK_DIM_2=1
#CHUNK_DIM_3=1
#============================================================
#    Async related settings
DELAYED_CLOSE_TIMESTEPS=2
IO_MEM_LIMIT=5000 K
ASYNC_MODE=EXP #EXP IMP NON
#============================================================
#    Output performance results to a CSV file
#CSV_FILE=perf_write_1d.csv
#
#FILE_PER_PROC=
```

General Settings

- **IO_OPERATION**: required, chose from **READ** and **WRITE**.
- **MEM_PATTERN**: required, chose from **CONTIG, INTERLEAVED** and **STRIDED**
- **FILE_PATTERN**: required, chose from **CONTIG**, and **STRIDED**
- **NUM_PARTICLES**: required, the number of particles that each rank needs to process, can be in exact numbers (12345) or in units (format like 16 K, 128 M and 256 G are supported, format like 16K, 128M, 256G is NOT supported).
- TIMESTEPS: required, the number of iterations
- **EMULATED_COMPUTE_TIME_PER_TIMESTEP**: required, must be with units (eg,. 10 s, 100 ms or 5000 us). In each iteration, the same amount of data will be written and the file size will increase correspondingly. After each iteration, the program sleeps for $EMULATED_COMPUTE_TIME_PER_TIMESTEP time to emulate the application computation.
- **NUM_DIMS**: required, the number of dimensions, valid values are 1, 2 and 3.
- **DIM_1, DIM_2, and DIM_3**: required, the dimensionality of the source data. Always set these parameters in ascending order, and set unused dimensions to 1, and remember that NUM_PARTICLES == DIM_1 * DIM_2 * DIM_3 MUST hold. For example, DIM_1=1024, DIM_2=256, DIM_3=1 is a valid setting for a 2D array when NUM_PARTICLES=262144 or NUM_PARTICLES=256 K, because 10242561 = 262144, which is 256 K.

Example for using multi-dimensional array data

- Using 2D as the example, 3D cases are similar, the file is generated with with 4 ranks, each rank write 8M elements, organized in a 4096 * 2048 array, in total it forms a (4 * 4096) * 2048 2D array. The file should be around 1GB.

Dimensionality part of the Config file:

```
NUM_DIMS=2
DIM_1=4096
DIM_2=2048
DIM_3=64        # Note: extra dimensions than specified by NUM_DIMS are ignored
```

Addtional Settings for READ (h5bench_read)

- **READ_OPTION**: required for IO_OPERATION=READ, not allowed for IO_OPERATION=WRITE.
  - FULL: read the whole file
  - PARTIAL: read the first $TO_READ_NUM_PARTICLES particles
  - STRIDED: read in streded pattern
- **TO_READ_NUM_PARTICLES**: required, the number for particles attempt to read.

Async Related Settings

- **ASYNC_MODE**: optional, the default is NON.
  - NON: the benchmark will run in synchronous mode.
  - EXP: enable the asynchronous mode. An installed async VOL connector and coresponding environment variables are required.
- **IO_MEM_LIMIT**: optional, the default is 0, requires **ASYNC_MODE=EXP**, only works in asynchronous mode. This is the memory threshold used to determine when to actually execute the IO operations. The actual IO operations (data read/write) will not be executed until the timesteps associated memory reachs the threshold, or the application run to the end.
- **DELAYED_CLOSE_TIMESTEPS**: optional, the default is 0. The groups and datasets associated to to the timesteps will be closed later for potential caching.

Compression Settings

- **COMPRESS**: YES or NO, optional. Only applicable for WRITE(h5bench_write), has no effect for READ. Used to enable compression, when enabled, chunk dimensions(CHUNK_DIM_1, CHUNK_DIM_2, CHUNK_DIM_3) are required. To enable parallel compression feature for VPIC, add following section to the config file, and make sure chunk dimension settings are compatible with the data dimensions: they must have the same rank of dimensions (eg,. 2D array dataset needs 2D chunk dimensions), and chunk dimension size cannot be greater than data dimension size.

```
COMPRESS=YES    # to enable parallel compression( chunking)
CHUNK_DIM_1=512 # chunk dimensions
CHUNK_DIM_2=256
CHUNK_DIM_3=1   # extra chunk dimension take no effects
```

Attention: 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.

Collective Operation Settings

- **COLLECTIVE_DATA**: optional, set to "YES" for collective data operations, otherwise and default (not set) cases for independent operations.
- **COLLECTIVE_METADATA**: optional, set to "YES" for collective metadata operations, otherwise and default (not set) cases for independent operations.

Other Settings

- **CSV_FILE**: optional CSV file output name, performance results will be print to the file and the standard output as well.

3.1.2 Supported Patterns

Attention: Not every pattern combination is covered, supported benchmark parameter settings are listed below.

Supported Write Patterns (h5bench_write): IO_OPERATION=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
```

Supported Read Patterns (h5bench_read): IO_OPERATION=READ

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

```
MEM_PATTERN=CONTIG, FILE_PATTERN=CONTIG, READ_OPTION=FULL, contiguously read through the␣
↪whole data file.
```

2 patterns for 1D read

```
MEM_PATTERN=CONTIG, FILE_PATTERN=CONTIG, READ_OPTION=PARTIAL, contiguously read the␣
↪first $TO_READ_NUM_PARTICLES elements.

MEM_PATTERN=CONTIG, FILE_PATTERN=STRIDED, READ_OPTION=STRIDED
```

3.1.3 Sample Settings

The following setting reads 2048 particles from 128 blocks in total, each block consists of the top 16 from every 64 elements. See HDF5 documentation for details of using strided access.

```
#   General settings
NUM_PARTICLES=16 M
TIMESTEPS=5
MULATED_COMPUTE_TIME_PER_TIMESTEP=1 s
#=================================================
#   Benchmark data dimensionality
NUM_DIMS=1
DIM_1=16777216
DIM_2=1
DIM_3=1
#=================================================
#   IO pattern settings
IO_OPERATION=READ
MEM_PATTERN=CONTIG
FILE_PATTERN=CONTIG
#=================================================
#    Options for IO_OPERATION=READ
READ_OPTION=PARTIAL # FULL PARTIAL STRIDED
TO_READ_NUM_PARTICLES=2048
#=================================================
#    Strided access parameters
STRIDE_SIZE=64
BLOCK_SIZE=16
BLOCK_CNT=128
```

For more examples, please find the config files and template.cfg in basic_io/sample_config/ directory.

3.1.4 To Run the h5bench_write and h5bench_read

Both h5bench_write and h5bench_read use the same command line arguments:

Single process run:

```
./h5bench_write sample_write_cc1d_es1.cfg my_data.h5
```

Parallel run (replace mpirun with your system provided command, for example, srun on Cori/NERSC and jsrun on Summit/OLCF):

```
mpirun -n 2 ./h5bench_write sample_write_cc1d_es1.cfg output_file
```

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"
```

3.1.5 Argobots in MacOS

If you're trying to run the benchmark in a MacOS and are getting segmentation fault (from ABT_thread_create), please try to set the following environment variable:

```
ABT_THREAD_STACKSIZE=100000 ./h5bench_write sample_write_cc1d_es1.cfg my_data.h5
```

3.1.6 Understanding the Output

The metadata and raw data operations are timed separately, and overserved time and 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
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
```

3.2 h5bench_exerciser

We modified this benchmark slightly so to be able to specify a file location that is writable. Except for the first argument $write_file_prefix, it's identical to the original one. Detailed README can be found int source code directory, the original can be found here https://xgitlab.cels.anl.gov/ExaHDF5/BuildAndTest/-/blob/master/Exerciser/README.md

Example run:

```
mpirun -n 8 ./h5bench_exerciser $write_file_prefix -numdims 2 --minels 8 8 --nsizes 3 --
↪bufmult 2 --dimranks 8 4
```

3.3 The Metadata Stress Test: h5bench_hdf5_iotest

This is the same benchmark as it's originally found at https://github.com/HDFGroup/hdf5-iotest. We modified this benchmark slightly so to be able to specify the config file location, everything else remains untouched.

Example run:

```
mpirun -n 4 ./h5bench_hdf5_iotest hdf5_iotest.ini
```

3.4 Streaming operation benchmark: h5bench_vl_stream_hl

This benchmark tests the performance of append operation. It supports two types of appends, FIXED and VLEN, represents fixed length data and variable length data respectively. Note: This benchmark doesn't run in parallel mode.

3.4.1 To run benchmarks

```
./h5bench_vl_stream_hl write_file_path FIXED/VLEN num_ops
```

Example runs:

```
./h5bench_vl_stream_hl here.dat FIXED 1000
./h5bench_vl_stream_hl here.dat VLEN 1000
```

CHAPTER

FOUR

HDF5 EXERCISER BENCHMARK

Authors:

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

Updates: December 13th 2018 (Version 2.0)

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.

4.1 Exerciser Overview

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.

4.1.1 Command-line Arguments (Options)

Required

- –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)

Optional

- –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>(one value for each dimension): 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>(one value for each dimension): 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:

- –dimranks <x>…<x>: (one value for each dimension) mpi-rank division in each dimension. Note that, if not set, decomposition will be in 1st dimension only

4.1.2 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.

If the user wants to loop through a range of buffer sizes, the –nsizes flag can be used to specify how many sizes measure, and the –bufmult flag can be used to specify the multiplication factor for each dimension between each loop. For example, if the user wanted to test 64x64, 128x128, and 256x256-element local datasets on 32 ranks, they could use the following command to run the code:

```
mpirun -np 32 ./hdf5Exerciser --numdims 2 --minels 8 8 --nsizes 3 --bufmult 2 2 --
↪dimranks 8 4
```

When executed for a single local-buffer size (default), the Exerciser output will look something like this:

```
useMetaDataCollectives: 0 addDerivedTypeDataset: 0 addAttributes: 0 useIndependentIO: 0␣
↪numDims: 1 useChunked: 0 rankShift: 4096
Metric      Bufsize   H5DWrite    RawWrBDWTH    H5Dread    RawRdBDWTH    Dataset     ␣
↪Group  Attribute    H5Fopen   H5Fclose   H5Fflush OtherClose
Min           32768   0.134616   3058.154823   0.191049   2534.613015   0.361010   0.
↪551608   0.000001   0.224550   0.127877   0.210821   0.000755
Med           32768   0.143874   3554.180478   0.191684   2670.829718   0.379858   0.
↪612309   0.000001   0.236735   0.132450   0.228889   0.000761
Max           32768   0.167421   3803.418460   0.202003   2679.939135   0.405620   0.
↪679779   0.000002   0.268622   0.138463   0.270188   0.000785
Avg           32768   0.146435   3506.598052   0.192068   2666.021346   0.379799   0.
↪616157   0.000001   0.237219   0.132410   0.233730   0.000763
Std           32768   0.008055    185.366133   0.002090     27.665058   0.010248   0.
↪026048   0.000000   0.008915   0.002650   0.017362   0.000006
```

Using NUM_ITERATIONS samples for each local buffer size (Bufsize), the minimum, median, maximum, average, and standard deviation of all metrics will be reported in distinct rows of the output. The Bufsize values are reported in **bytes**, while the RawWrBDWTH and RawRdBDWTH are in **MB/s**, and all other metrics are in **seconds**.

4.2 Building Exerciser

H5bench's make process builds the h5bench_exerciser.

In case, Exerciser needs to be built separately, given the path to a parallel HDF5 installation, building it is straightforward. The following Makefile can be used as a reference:

```
default: hdf5Exerciser

HDF5_INSTALL_DIR=/Users/rzamora/hdf5-install

```

(continues on next page)

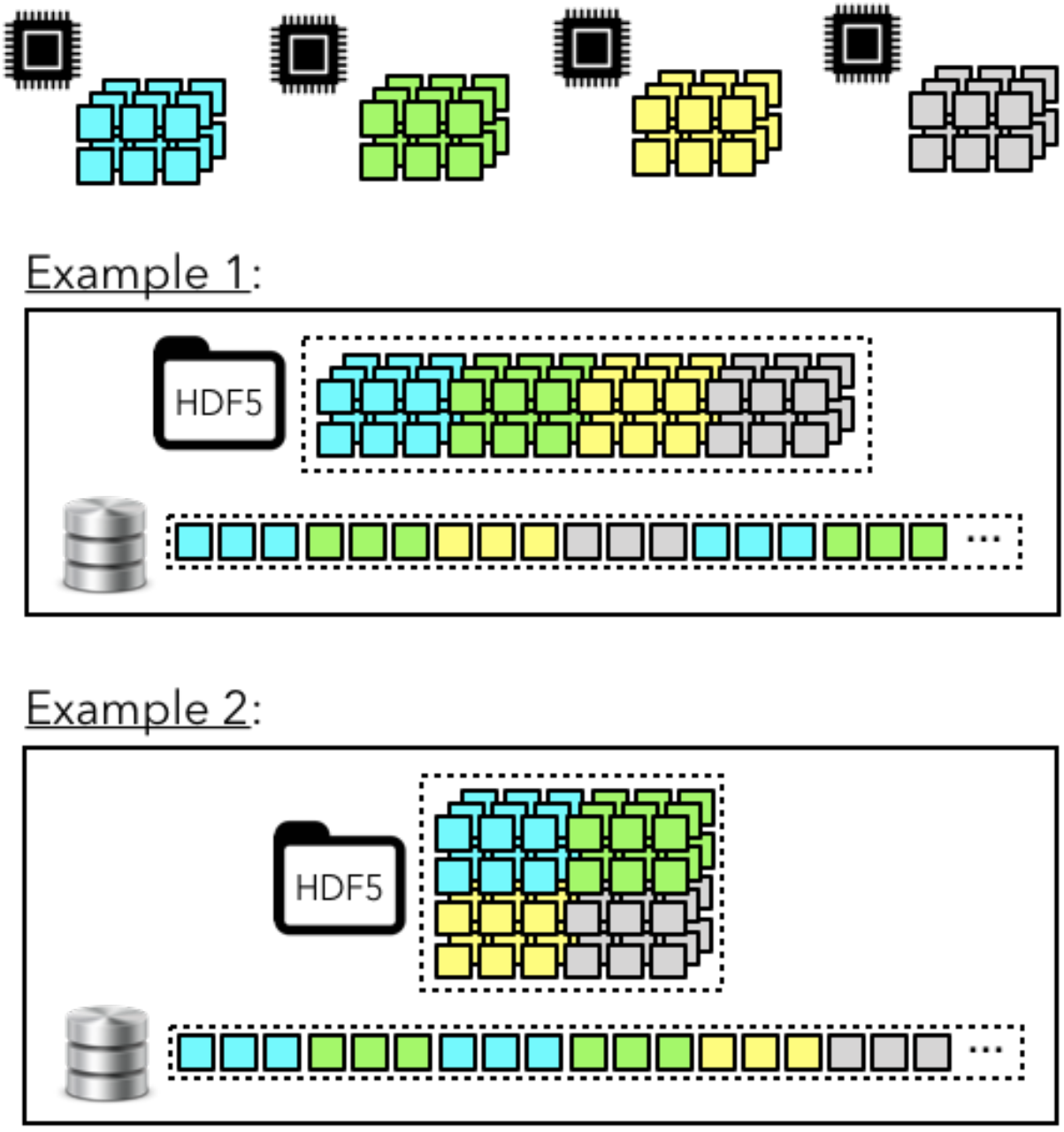

Fig. 1: Fig. 1 - Illustration of different local-to-global dataset mapping options.

(continued from previous page)

```
h5bench_exerciser.o: h5bench_exerciser.c
mpicc  -c -g -DMETACOLOK -I${HDF5_INSTALL_DIR}/include  h5bench_exerciser.c -o h5bench_
↪exerciser.o

hdf5Exerciser: h5bench_exerciser.o
mpicc h5bench_exerciser.o -o hdf5Exerciser  -L${HDF5_INSTALL_DIR}/lib -lhdf5 -lz

clean:
rm -f h5bench_exerciser.o
rm -f hdf5Exerciser
```

For more-detailed instructions of how to build and run both HDF5 and the exerciser on specific machines (at ALCF), see the Exerciser/BGQ/VESTA_XL and Exerciser/BGQ/THETA directories of this repository.

CHAPTER

FIVE

H5BENCH_PATTERNS

Here you can find some sample configuration files for h5bench patterns:

5.1 sample_2d_compressed.cfg

```
# this is a comment
# Benchmark mode can only be one of these: CC/CI/IC/II/CC2D/CI2D/IC2D/II2D/CC2D/CC3D
IO_OPERATION=READ
MEM_PATTERN=CONTIG
FILE_PATTERN=CONTIG
READ_OPTION=FULL
NUM_DIMS=2
NUM_PARTICLES=8 M
TIMESTEPS=5
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1 s
DIM_1=4096
DIM_2=2048
DIM_3=1
COMPRESS=YES  # to enable parallel compression(chunking)
CHUNK_DIM_1=512  # chunk dimensions
CHUNK_DIM_2=256
CHUNK_DIM_3=1
```

5.2 sample_read_cc1d.cfg

```
# this is a comment
# Benchmark mode can only be one of these: CC/CI/IC/II/CC2D/CI2D/IC2D/II2D/CC2D/CC3D
IO_OPERATION=READ
READ_OPTION=FULL
NUM_DIMS=1
NUM_PARTICLES=8 M
TIMESTEPS=5
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1 s
DIM_1=8 M
DIM_2=1
DIM_3=1
```

5.3 sample_read_cc1d_es1.cfg

```
# this is a comment
IO_OPERATION=READ
TO_READ_NUM_PARTICLES=16777216
READ_OPTION=READ_FULL
MEM_PATTERN=CONTIG
FILE_PATTERN=CONTIG
TIMESTEPS=5
DELAYED_CLOSE_TIMESTEPS=2
COLLECTIVE_DATA=NO #Optional, default for NO.
COLLECTIVE_METADATA=NO #Optional, default for NO.
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1 s
NUM_DIMS=1
DIM_1=16777216 # 16777216, 8388608
DIM_2=1
DIM_3=1
IO_MEM_LIMIT=1 G
ASYNC_MODE=EXP  #NON
#CSV_FILE=perf_read_1d.csv
#==================================
```

5.4 sample_read_cc2d.cfg

```
# this is a comment
# Benchmark mode can only be one of these: CC/CI/IC/II/CC2D/CI2D/IC2D/II2D/CC2D/CC3D
IO_OPERATION=READ
MEM_PATTERN=CONTIG
FILE_PATTERN=CONTIG
READ_OPTION=FULL
NUM_DIMS=2
NUM_PARTICLES=8 M
TIMESTEPS=5
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1 s
DIM_1=2048
DIM_2=4096
DIM_3=1
```

5.5 sample_read_strided.cfg

```
# this is a comment
# Benchmark mode can only be one of these: CC/CI/IC/II/CC2D/CI2D/IC2D/II2D/CC2D/CC3D
IO_OPERATION=READ
TO_READ_NUM_PARTICLES=16777216
READ_OPTION=FULL
MEM_PATTERN=CONTIG
FILE_PATTERN=STRIDED
TIMESTEPS=5
```

(continues on next page)

(continued from previous page)

```
DELAYED_CLOSE_TIMESTEPS=10
COLLECTIVE_DATA=NO #Optional, default for NO.
COLLECTIVE_METADATA=NO #Optional, default for NO.
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1 s
NUM_DIMS=1
DIM_1=16777216 # 16777216, 8388608
DIM_2=1
DIM_3=1
STRIDE_SIZE=64
BLOCK_SIZE=16
BLOCK_CNT=128
ASYNC_MODE=NO  #NON
CSV_FILE=perf_read_1d.csv
#==================================
```

5.6 sample_write_cc1d.cfg

```
# this is a comment
# Benchmark mode can only be one of these: CC/CI/IC/II/CC2D/CI2D/IC2D/II2D/CC2D/CC3D
# Template cof include all options
IO_OPERATION=WRITE
MEM_PATTERN=CONTIG
FILE_PATTERN=CONTIG
NUM_PARTICLES=16 M #16 K/G
TIMESTEPS=5
#IO_OPERATION=READ #WRITE
#MEM_PATTERN=CONTIG #INTERLEAVED STRIDED
#FILE_PATTERN=CONTIG #STRIDED
DELAYED_CLOSE_TIMESTEPS=2
COLLECTIVE_DATA=NO #Optional, default for NO.
COLLECTIVE_METADATA=NO #Optional, default for NO.
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1 s #1 ms, 1 min
NUM_DIMS=1
DIM_1=16777216 #16777216 # 16777216, 8388608
DIM_2=1
DIM_3=1
ASYNC_MODE=NON #EXP #ASYNC_IMP ASYNC_NON ASYNC_EXP
CSV_FILE=perf_write_1d.csv
#==========================
#WRITE_PATTERN=CC
```

5.7 sample_write_cc1d_es1.cfg

```
# this is a comment
IO_OPERATION=WRITE
MEM_PATTERN=CONTIG
FILE_PATTERN=CONTIG
NUM_PARTICLES=16 M #K, M, G
TIMESTEPS=5
DELAYED_CLOSE_TIMESTEPS=2
COLLECTIVE_DATA=NO
#Optional, default for NO.
COLLECTIVE_METADATA=NO
#Optional, default for NO.
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1 s
#1 ms, 1 min
NUM_DIMS=1
DIM_1=16777216
#16777216 # 16777216, 8388608
DIM_2=1
DIM_3=1
IO_MEM_LIMIT=1 G
#ASYNC_MODE=ASYNC_EXP
ASYNC_MODE=EXP #IMP NON EXP
#CSV_FILE=perf_write_1d.csv
#============================
#WRITE_PATTERN=CC
```

5.8 sample_write_cc1d_fileperproc.cfg

```
# this is a comment
# Benchmark mode can only be one of these: CC/CI/IC/II/CC2D/CI2D/IC2D/II2D/CC2D/CC3D
WRITE_PATTERN=CC
PARTICLE_CNT_M=8
TIME_STEPS_CNT=1
DATA_COLL=NO #Optional, default for NO.
META_COLL=NO #Optional, default for NO.
SLEEP_TIME=1
DIM_1=8388608
DIM_2=1
DIM_3=1
ASYNC_MODE=ASYNC_NON
CSV_FILE=perf_write_1d.csv
FILE_PER_PROC=YES #Optional, default is NO.
```

5.9 sample_write_cc2d.cfg

```
# this is a comment
# Benchmark mode can only be one of these: CC/CI/IC/II/CC2D/CI2D/IC2D/II2D/CC2D/CC3D
# Template cof include all options
IO_OPERATION=WRITE
MEM_PATTERN=CONTIG
FILE_PATTERN=CONTIG
NUM_PARTICLES=16 M #16 K/G
TIMESTEPS=5
#IO_OPERATION=READ #WRITE
#MEM_PATTERN=CONTIG #INTERLEAVED STRIDED
#FILE_PATTERN=CONTIG #STRIDED
DELAYED_CLOSE_TIMESTEPS=2
COLLECTIVE_DATA=NO #Optional, default for NO.
COLLECTIVE_METADATA=NO #Optional, default for NO.
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1 s #1 ms, 1 min
NUM_DIMS=2
DIM_1=4096 #16777216 # 16777216, 8388608
DIM_2=4096
DIM_3=1
ASYNC_MODE=NON #EXP #ASYNC_IMP ASYNC_NON ASYNC_EXP
CSV_FILE=perf_write_1d.csv
#============================
#WRITE_PATTERN=CC
```

5.10 sample_write_strided.cfg

```
# this is a comment
# Benchmark mode can only be one of these: CC/CI/IC/II/CC2D/CI2D/IC2D/II2D/CC2D/CC3D
WRITE_PATTERN=CC
NUM_PARTICLES=16
TIMESTEPS=1
COLLECTIVE_DATA=NO #Optional, default for NO.
COLLECTIVE_METADATA=NO #Optional, default for NO.
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1
DIM_1=8388608
DIM_2=1
DIM_3=1
STRIDE_SIZE=2
BLOCK_SIZE=2
BLOCK_CNT=1048576
```

5.11 template.cfg

```
#=============================================================
#   General settings
NUM_PARTICLES=16 M # 16 K  16777216
TIMESTEPS=5
EMULATED_COMPUTE_TIME_PER_TIMESTEP=1 s #1 ms, 1 min
#=============================================================
#   Benchmark data dimensionality
NUM_DIMS=1
DIM_1=16777216 # 16777216, 16 M
DIM_2=1
DIM_3=1
#=============================================================
#   IO pattern settings
IO_OPERATION=READ # WRITE
MEM_PATTERN=CONTIG # INTERLEAVED STRIDED
FILE_PATTERN=CONTIG # STRIDED
#=============================================================
#    Options for IO_OPERATION=READ
READ_OPTION=FULL # PARTIAL STRIDED
TO_READ_NUM_PARTICLES=4 M
#=============================================================
#    Strided access parameters
#STRIDE_SIZE=
#BLOCK_SIZE=
#BLOCK_CNT=
#=============================================================
# Collective data/metadata settings
#COLLECTIVE_DATA=NO #Optional, default for NO.
#COLLECTIVE_METADATA=NO #Optional, default for NO.
#=============================================================
#    Compression, optional, default is NO.
#COMPRESS=NO
#CHUNK_DIM_1=1
#CHUNK_DIM_2=1
#CHUNK_DIM_3=1
#=============================================================
#    Async related settings
DELAYED_CLOSE_TIMESTEPS=2
IO_MEM_LIMIT=5000 K
ASYNC_MODE=EXP #EXP NON
#=============================================================
#    Output performance results to a CSV file
#CSV_FILE=perf_write_1d.csv
#
#FILE_PER_PROC=
```

CHAPTER

SIX

COPYRIGHT

CHAPTER

SEVEN

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.

CHAPTER

EIGHT

INDICES AND TABLES

- genindex
- modindex
- search