MHM User Guide

Version 2.1.0.2 (01/31/2023)

Acknowledgments

This work was supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

MetaHipMer (*MHM*) is a *de novo* metagenome short-read assembler. This is version 2 (MHM2), which is written in UPC++, CUDA and HIP, and runs efficiently on both single servers and on multinode supercomputers, where it can scale up to coassemble terabase-sized metagenomes. More information about MetaHipMer can be found under the ExaBiome Project of the Exascale Computing Project and in several publications:

- E. Georganas et al., "Extreme Scale De Novo Metagenome Assembly," SC18: International Conference for High Performance Computing, Networking, Storage and Analysis, Dallas, TX, USA, 2018, pp. 122-13.
- Hofmeyr, S., Egan, R., Georganas, E. et al. Terabase-scale metagenome coassembly with MetaHipMer. Sci Rep 10, 10689 (2020).
- Awan, M.G., Deslippe, J., Buluc, A. et al. ADEPT: a domain independent sequence alignment strategy for gpu architectures. BMC Bioinformatics 21, 406 (2020).
- Muaaz Awan, Steven Hofmeyr, Rob Egan et al. "Accelerating large scale de novo metagenome assembly using GPUs.", SC 2021

The quality of MetaHipMer assemblies is comparable with other leading metagenome assemblers, as documented in the results of the CAMI2 competition, where MetaHipMer was rated first for quality in two out of three datasets, and second in the third dataset:

- F. Meyer et al., "Critical Assessment of Metagenome Interpretation: the second round of challenges", Nature Methods volume 19, pages429–440 (2022)

MHM2 is developed and maintained by the Exabiome Project at Lawrence Berkeley National Laboratory, and is supported by the Exascale Computing Project (17-SC-20-SC), a collaborative effort of the U.S. Department of Energy Office of Science and the National Nuclear Security Administration.

Building and Installing

MHM2 depends on UPC++, with the C++17 standard, and CMake. GPU builds require CUDA and/or HIP.

A script, `build.sh`, is provided for building and installing MHM2.

Before building MHM2, ensure that either the UPC++ compiler wrapper, `upcxx` is in your `PATH`, or set the `MHM2_BUILD_ENV` environment variable to point to a script that loads the appropriate environment, for example, on NERSC's Perlmutter supercomputer, you would set the following for the gnu compiler on the KNL partition:

```
export MHM2_BUILD_ENV=contrib/environments/perlmutter/gnu.sh
```

There are several scripts provided for different build choices on NERC's and OLCF's systems, in directories that start with `contrib/environments`. You do not need to use any scripts such as these when building on a Linux server, although you may want to create your own when setting up the build. On NERSC and OLCF we recommend using the gnu (`contrib/environments/*/gnu.sh`) environments. Building with Intel is very slow.

To build a release version (optimized for performance), execute:

```
./build.sh Release
```

Alternatively, you can build a debug version with:

```
./build.sh Debug
```

This will capture a great deal of information useful for debugging but will run a lot slower (up to 5x slower at scale on multiple nodes).

An alternative to the pure debug version is the "release" debug version, which still captures a reasonable amount of debugging information, but is a lot faster (although still up to 2x slower than the release version):

```
./build.sh RelWithDebInfo
```

The `./build.sh` script will install the binaries by default into the `install/bin` subdirectory in the repository root directory. To set a different install directory, set the environment variable `MHM2_INSTALL_PATH`, e.g.:

```
MHM2_INSTALL_PATH=/usr/local/share/mhm2 ./build.sh Release
```

Once MHM2 has been built once, you can rebuild with

```
./build.sh
```

and it will build using the previously chosen setting (`Release`, `Debug`, or `RelWithDebInfo`).

You can also run

```
./build.sh clean
```

to start from scratch. If you run this, then the next call to `build.sh` should be with one of the three configuration settings.

By default, the build occurs within the root of the repository, in a subdirectory called `.build`. This is created automatically by the `build.sh` script.

The MHM2 build uses `cmake`, which you can call directly, instead of through the `build.sh` script, e.g.:

```
mkdir -p .build
cd .build
cmake -DCMAKE_INSTALL_PREFIX=path-to-install ..
make -j all install
```

Consult the `build.sh` script to see how it executes these commands.

You'll need to first set the environment, e.g.:

```
source contrib/environments/perlmutter/gnu.sh
```

If you see an error message when building like the following:

```
include could not find load file: GetGitVersion
```

Then you have probably not cloned the git submodules. You need to execute the following from the root directory:

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

Running

To execute MHM2, run the `mhm2.py` script located at `install/bin`. Most parameters have sensible defaults, so it is possible to run with only the read FASTQ files specified, e.g. to run with two interleaved reads files, `lib1.fastq` and `lib2.fastq`, you could execute:

```
mhm2.py -r lib1.fastq,lib2.fastq
```

A list of all the command line options can be found by running with `-h`. Because `mhm2.py` is a python script that wraps the UPC++ binary, `mhm2`, there will be two levels of options, one from the python script, and one from the binary. Some of the options have a short form (a single dash with a single character) and a long form (starting with a double-dash). In the options described below, where both a short form and a long form exist, they are separated by a comma. The type of the option is indicated as one of `STRING` (a string of characters), `INT` (an integer), `FLOAT` (a floating point value) or `BOOL` (a boolean flag). For `BOOL`, the option can be given as `true`, `false`, `yes`, `no`, `0`, `1`, or omitted altogether, in which case the option will be `true`, and if an option is specfied, the `=` must be used, e.g.

```
mhm2.py --checkpoint=false
```

By default, the run will generate files in a specific output directory (see the `--output` option below). At a minimum, this will include the following files:

- `final_assembly.fasta`: the contigs for the assembly, in FASTA format.
- `mhm2.log`: a log file containing details about the run, including various quality statistics, details about the assembly process and timing information.
- `mhm2.config`: a configuration file containing all the non-default options used for the run.
- `per_thread`: a subdirectory containing per-process files that record memory usage and debugging information in Debug mode.

In addition, many more files may be generated according to which command-line options are specified. These are described in detail below where relevant.

The `mhm2` binary can be executed directly using `upcxx-run`, `srun` or another suitable launcher. Generally we recommend using `mhm2.py`, because it takes care of many facets of starting the executable in a given environment and provides additional functionality, e.g. automatically restarting on errors, easily enabling communication tracing, etc.

Basic options

These are the most commonly used options.

The input files of reads are specified with either `-r`, `-p`, or `-u`. At least one of these options must be specified. When running on a Lustre file system (such as on OLCF's Frontier), it is recommended that all input files be striped to ensure adequate I/O performance. Usually this means first striping a directory and then moving files into it, e.g. for a file `reads.fastq`:

```
mkdir data
lfs setstripe -c 72 data
mv reads.fastq data
```

`-r, --reads STRING,STRING,...`

A collection of names of files containing interleaved paired reads in FASTQ format. Multiple files must be comma-separated, or can be separated by spaces. For paired reads in separate files, use the `-p` option. For unpaired reads, use the `-u` option. Long lists of read files can be set in a configuration file and loaded with the `--config` option, to avoid having to type them in on the command line.

`-p, --paired-reads STRING,STRING,...`

A collection of names of files containing separate paired reads in FASTQ format. Multiple files must be comma-separated, or can be separated by spaces. For each library, the file containing the reads for the first pairs must be followed by the file containing the reads for the second pairs, e.g. for two libraries with separate paired reads files `lib1_1.fastq`, `lib1_2.fastq` and `lib2_1.fastq`, `lib2_2.fastq`, the option should be specified as:

```
-p lib1_1.fastq,lib1_2.fastq,lib2_1.fastq,lib2_2.fastq
```

This option only supports reads where each pair of reads has the same sequence length, usually only seen in raw reads. For support of trimmed reads of possibly different lengths, first interleave the files and then call with the `-r` option. The separate files can be interleaved with `reformat.sh` from bbtools.

`-u, --unpaired-reads STRING,STRING,...`

A collection of names of files containing unpaired reads in FASTQ format. Multiple files must be comma-separated, or can be separated by spaces.

`--adapter-refs STRING`

A file containing adapter sequences in the FASTA format. If specified, it will be used to trim out all adapters when the input reads are first loaded. Two files containing adapter sequences are provided in the `contrib`

directory: `adapters_no_transposase.fa` and `all_adapters.fa.gz`. The latter must be `gunzip`ped before it can be used.

`-i, --insert INT:INT`

The insert size for paired reads. The first integer is the average insert size for the paired reads and the second integer is the standard deviation of the insert sizes. MHM2 will automatically attempt to compute these values so this parameter is usually not necessary. However, there are certain cases where it may be useful, for example, if MHM2 prints a warning about being unable to compute the insert size because of the nature of the reads, or if only doing scaffolding. MHM2 will also compare its computed value to any option set on the command line and print a warning if the two differ significantly; this is useful for confirming assumptions about the insert sise distribution.

`-k, --kmer-lens INT,INT,...`

The k-mer lengths used for the contigging rounds. MHM2 performs one or more contigging rounds, each of which performs k-mer counting, followed by a deBruijn graph traversal, then alignment and local assembly to extend the contigs. Typically, multiple rounds are used with increasing values of k; the shorter values are useful for low abundance genomes, whereas the longer k values are useful for resolving repeats. This option defaults to `-k 21,33,55,77,99`, which is fine for reads of length 150. For shorter or longer reads, it may be a good idea to adjust these values, for example, for reads of length 101, a better set is usually `-k 21,33,47,63`. Also, each round of contigging takes time, so the overall assembly time an be reduced by reducing the number of rounds, although this will likely reduce the quality of the final assembly.

`-s, --scaff-kmer-lens INT,INT,...`

The k-mer lengths used for the scaffolding rounds. In MHM2, the contigging rounds are followed by one or more scaffolding rounds. These rounds usually proceed from a high k to a low one, i.e. the reverse ordering of contigging. This option defaults to `-s 99,33`. The first value should always be set to the final `k` used in contigging, e.g. for reads of length 101 with parameter `-k 21,33,47,63`, the scaffolding values could be `-s 63,33`. More rounds may improve contiguity but will likely increase misassemblies. To disable scaffolding altogether, set this value to 0, i.e. `-s 0`.

`--min-ctg-print-len INT`

The minimum length for contigs to be included in the final assembly, `final_assembly.fasta`. This defaults to 500.

`-o, --output STRING`

The name for the output directory. If not specified, it will be set to a default value of the following form:

`mhm2-run-<READS_FNAME1>-n<PROCS>-N<NODES>-YYMMDDhhmmss-<JOBID>`

where `<READS_FNAME1>` is the name of the first reads file, `PROCS` is the number of processes and `NODES` is the number of nodes. Following this is the date and time when the run was started: `YY` is the last two digits of the year, `MM` is the number of the month, `DD` is the day of the month, `hh` is the hour of day, `mm` is the minute and `ss` is the second. Be warned that if two runs are started at exactly the same time, with the same parameters, then with the default values, they could both end up running in the same output directory, which will lead to corrupted results.

If the output directory is created by MHM2 (either as the default or when passed as a parameter), it will automatically be striped in the most effective way on a Lustre filesystem. If using a pre-existing directory that was not created by MHM2, the user should ensure that on Lustre filesystems it is adequately striped.

If the output directory already exists, files produced by a previous run of MHM2 may be overwritten, depending on whether or not this is a restart of a previous run. If there is an existing log file (`mhm2.log`), it will be renamed with the date appended before the new one is written as `mhm2.log`, so log information about previous runs will always be retained.

`--checkpoint BOOL`

Checkpoint runs. If set to true, this will checkpoint the run by saving intermediate files that can later be used to restart the run (see the `--restart` option below). The intermediate files are FASTA files of contigs, and they are saved at the end of each contigging round (`contigs-<k>.fasta`) and at the end of each scaffolding round (`scaff-contigs-<k>.fasta`), where the `<k>` value is the *k*-mer size for that round. Checkpointing is on by default and can be disabled by passing `--checkpoint=false`.

`--restart BOOL`

Restart a previous incomplete run. If set to true, MHM2 will attempt to restart a run from an existing directory. The output directory option must be specified and must contain a previous checkpointed run. The restart will use the same options as the previous run, and will load the most recent checkpointed contigs file in order to resume. This defaults to false.

`--post-asm-align BOOL`

Perform alignment of reads to final assembly after assembly has completed. If set to true. MHM2 will align the original reads to the final assembly and report the results in a file, `final_assembly.sam`, in SAM format. This defaults to false.

`--post-asm-abd BOOL`

Compute contig abundances after assembly has completed. If set to true, MHM2 will compute the abundances (depths) for the contigs in the final assembly and write the results to the file, `final_assembly_depths.txt`. The format of this file is the same as that used by MetaBAT, and so can be used together with the `final_assembly.fasta` for post-assembly binning, e.g.:

```
metabat2 -i final_assembly.fasta -a final_assembly_depths.txt -o bins_dir/bin
```

This defaults to false.

`--post-asm-only BOOL`

Perform only post-assembly operations. If set to true, this requires an existing directory containing a full run (i.e. with a `final_assembly.fasta` file), and it will execute any specified post-assembly options (`--post-asm-align`, `--post-asm-abd`) on that assembly without any other steps. This provides a convenient means to run alignment and/or abundance calculations on an already completed assembly. By default this post-assembly analysis will use the `final_assembly.fasta` file in the output directory, but any FASTA file could be used, including those not generated by MHM2 (see the `--contigs` in the advanced options section below). This defaults to false.

`--write-gfa BOOL`

Produce an assembly graph in the GF2 format. If set to true, MHM2 will output an assembly graph in the GFA2 format in the file, `final_assembly.gfa`. This represents the assembly graph formed by aligning the reads to the final contigs and using those alignments to infer edges between the contigs. This defaults to false.

`-Q, --quality-offset INT`

The *phred* encoding offset. In most cases, MHM2 will be able to detect this offset from analyzing the reads file, so it usually does not need to be explicitly set.

`--progress BOOL`

Display progress indicators during a run. If true, many time-consuming stages will be shown updating with a simple progress bar. The progress bar output will not be written into the log file, `mhm2.log`. This defaults to false.

`-v, --verbose BOOL`

Verbose output. If true, MMHM2 will produce verbose output, which prints out a lot of additional information about the timing of the run and the various computations that are being performed. This defaults to false. All of the information seen in verbose mode will always be written to the log file, `mhm2.log`. This defaults to false.

`--config STRING`

Use a config file for the parameters. If this is specified, the options will be loaded from the named config file. The file is a plain text file of the format:

```
key = value
```

where `key` is the name of an option and `value` is the value of the option. All blank lines and lines beginning with a semi-colon will be ignored. When the config file is not specified as an option, MHM2 always writes out all of the non-default options to the file `mhm2.config` in the output directory. Even when options are loaded from a config file, they can still be overridden by options on the command line. For example, if the config file, `test.config`, contains the line:

```
k = 21,33,55,77,99
```

but the command line is:

```
mhm2.py --config test.config -k 45,63
```

then MHM2 will run with k-mer lengths of 45, 63.

Advanced options

These are additional options for tuning performance or the quality of the output, selecting precisely how to restart a run, or for additonal debugging information. Most users will not need any of these options.

Restarting runs

Although the `--restart` option provides for simple restarts of previous runs, it is possible to restart at very specific points, with different options from those of the original run, e.g. restarting scaffolding with different k-mer values, set using the `-s` option.

The relevant options are listed below.

`-c, --contigs STRING`

The file name containing contigs in FASTA format that are to be used as the most recent checkpoint for a restart. Any contigs file generated during a checkpointed run can be used, so it is possible to restart at any stage. It is also possible to specify *any* FASTA file if running only post-assembly analysis (`--post-asm-only`).

`--max-kmer-len INT`

The maximum k-mer length that was previously used in contigging. This is usually derived from the `-k` parameter, and so only needs to be specified if the restart will run scaffolding rounds only. For example, the following command will restart after the scaffolding round with `k=99` and will run run two more scaffolding rounds with `k=55` and `k=21`:

```
mhm2.py -o outdir -r reads.fq -c scaff-contigs-99.fasta --max-kmer-len 99 -s 55,21
```

`--prev-kmer-len INT`

The k-mer length in the previous contigging round. Only needed if restarting in contigging, e.g.

```
mhm2.py -o outdir -r reads.fq -c contigs-77.fasta --max-prev-kmer-len 55
```

Tuning assembly quality

There are several additonal options for adjusting the quality of the final assembly, apart from the k-mer values specified for the contigging and scaffolding rounds, as described earlier.

`--break-scaff-Ns INT`

The number of Ns allowed in a gap before the scaffold is broken into two. The default is 10.

`--min-depth-thres INT`

The minimum depth (abundance) for a k-mer to be considered for the deBruijn graph traversal. This defaults to 2. Increasing it can reduce errors at the cost of reduced contiguity and genome discovery.

`--optimize STRING`

Adjust the trade-off in assembly quality between errors and contiguity. There are three settings that can be used: `contiguity` (improve contiguity at the cost of increased errors), `correctness` (reduce errors at the cost of contiguity), and `default` (the default setting which tries to balance the two).

Adjusting performance and memory usage

There are several options that adjust the trade-off between the memory used and the time taken, or that influence the performance in different ways on different platforms. These usually do not have to be adjusted.

`--max-kmer-store INT`

The maximum size per process in MB for the aggregation of k-mers during k-mer analysis. This defaults to 1% of available memory. Higher values use more memory, but can potentially result in faster computation.

`--max-rpcs-in-flight INT`

The maximum number of remote procedure calls (RPCs) outstanding at any given time. The default is 100. Reducing this will reduce memory usage but could increase running time. If set to 0, there are no limits on the outstanding RPCs.

`--max-worker-threads INT`

The maximum number of background worker threads. These threads are used for a limited number of background computation tasks. The default value is three and should not need to be adjusted.

`--pin STRING`

Restrict the hardware contexts that processes can run on. There are five options: `cpu` (restrict each process to a single logical CPU); `core` (restrict each process to a core); `numa` (restrict each process to a NUMA domain); `rr_numa` (restrict each process to a NUMA domain in a round robin manner; and `none` (don't restrict the processes). The default is `cpu`.

`--sequencing-depth INT`

The expected average sequencing depth. This value is used to estimate the memory requirements for unique k-mers in the first round of contigging. It may only need to be adjusted when memory is scarce and there is a need to better balance initial memory allocations.

`--shared-heap INT`

Set the shared heap size used by the UPC++ runtime, as a percentage of the available memory. This defaults to 10% and should not need to be adjusted. If MHM2 fails with `upcxx::bad_shared_alloc` messages, then this value should be increased.

Miscellaneous

`--shuffle-reads BOOL`

Shuffle reads to improve locality. This defaults to true, and results in greatly enhanced performance. The only reason to disable it is when testing or evaluating the impact of read shuffling.

`--use-qf BOOL`

Use the TCF filter to reduce memory when running on GPUs. This option will reduce peak GPU memory requirements by about 50%, with a performance impact of less than one percent. The only reason to disable it is for debugging or evaluating the impact of the TCF.

`--dump-merged BOOL`

Write merged FASTQ input files to the output directory. The file will be named `<READ_FILE_NAME>-merged.fastq`. The default is false.

`--dump-kmers BOOL`

Write k-mers to files in the output directory. The k-mers are written in each contigging stage immediately after k-mer counting. The files will be written on a per process basis into the `per_rank` subdirectory, named `kmers-<k>.txt.gz`, where `k` is the value for the contigging round. The default is false.

`--subsample-pct INT`

Percentage of input read files to use in the assembly. The value is from 0 to 100. This option enables to user to test a dataset with a smaller part of it, e.g. 10%. All of the input files will be reduced to the percentage specified.

`--procs INT`

Set the number of processes used when running MHM2. By default, MHM2 automatically detects the number of available cores and runs with one process per core. This setting allows a user to run MHM2 on a subset of available processors, which may be desirable if running on a server that is running other applications.

`--nodes INT`

Set the number of nodes on which to execute MHM2. By default, MHM2 will use the total available number of nodes from the job launch. On rare occasions it may be desirable to explicitly set this through the MHM2 launch, and not during the external job launch.

`--gasnet-stats`

Collect GASNet communication statistics. This option should be run with either a `Debug` or a `RelWithDebInfo` build. It will produce a summary of communication statistics for each stage, and write the summary to the log file and to stdout. Enabling this will further increase the runtime.

`--gasnet-trace`

Enable GASNet tracing. This must be run with a `Debug` or `RelWithDebInfo` build. This will produce one file per process `P`, called `trace_<P>.txt`. For more about what data are collected, consult the GASNet documentation, in the section about GASNet tracing and statistical collection. The trace mask is set to a comprensive set of default, i.e. setting the GASNet environment variable: `GASNET_TRACEMASK="GPWBNIH"`. This can be overriden by explicitly setting that environment variable.

`--preproc STRING`

Preprocessing commands. This is a comma separated list containing preprocesses and/or options (e.g. `valgrind --leak-check=full`), or additional options to be passed to `upcxx-run`.

`--binary STRING`

The name of the binary file for executing MHM2. This defaults to `mhm2`. This option can be used to run different binaries from the same script, such as builds with and without GPU support.