MHM User Guide

Version 2.0 (09/30/2020)

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 document is for version 2 (MHM2), which is written entirely in UPC++ and runs efficiently on both single servers and on multinode supercomputers, where it can scale up to coassemble terabase-sized metagenomes. More information about MHM can be found in:

- 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).

Building and Installing

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

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 Cori supercomputer, you would set the following for the gnu compiler on the KNL partition:

```
export MHM2_BUILD_ENV=contrib/environments/cori-knl/gnu.sh
```

There are several scripts provided for different build choices on NERC's Cori computer and the Summit supercomputer at OLCF, 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 Cori, we recommend using either the gnu (`contrib/environments/cori-*/gnu.sh`) or cray (`contrib/environments/cori-*/cray.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
```

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

```
source contrib/environments/cori-knl/gnu.sh
```

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`. Some of these have a short form (a single dash with a single character and a long form, starting with a double-dash). In the list 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 the 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 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.

Basic options

These are the most commonly used options.

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

A collection of names of files containing reads in FASTQ format. Multiple files must be comma-separated, without spaces. If the files contain paired-reads, these should be interleaved. For paired reads in separate files, use the `-p` option. Either this or the `-p` option are required. For long lists of read files, they can be set in a configuration file, to avoid having to type them in on the command line.

When running on a Lustre file system (such as on NERSC's Cori), it is recommended that all input files be striped to ensure adequate I/O performance. Usually this means striping a directory and then moving files into it, e.g. for a file `reads.fastq`:

```
mkdir data
lfs setstripe -c -1 data
mv reads.fastq data
```

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

File names for paired reads in FASTQ format contained in separate files. 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`

-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. Also, each round of contigging takes time, so the overall assembly time can 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`. 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`

where `<READS_FNAME1>` is the name of the first reads file, `PROCS` is the number of processes and `NODES` is the number of nodes. The final part is the date when the run was started: `YY` is 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, 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 can 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 before the new one is written as `mhm2.log`, so log information about previous runs will always be retained.

--checkpoint BOOL

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 contigs at the end of each contigging round are saved in a FASTA file, called `contigs-<k>.fasta`, where k is the k-mer value of the contigging round. At the end of each scaffolding round, the contigs will be saved in a FASTA file, `scaff-contigs-<k>.fasta`, where k is the k-mer value of the scaffolding round. Checkpointing is on by default and can be disabled by passing `--checkpoint=false`.

--restart BOOL

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

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

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

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` or `--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` option in the advanced options section below). This defaults to false.

--write-gfa BOOL

If set to true, MHM2 will output an assembly graph in the GFA2 format in 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

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`.

-v, --verbose BOOL

If true, MMHM2 will produce verbose output, which prints out a lot of additional information about 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`.

--config CONFIG_FILE_NAME

If this is specified, the options will be loaded from a config file. The file is 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 the options in the file `mhm2.config`.

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 and 63.

Advanced options

These are additional options for tuning performance or the quality of the output, or selecting precisely how to restart a run. 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 new options, 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.

`--max-kmer-len INT`

The maximum k-mer length used in contigging. This is usually derived from the parameters, and so only needs to be specified if the restart is only scaffolding rounds, and no contigging rounds, e.g.

```
mhm2.py -o outdir -r reads.fq -c scaff-contigs-33.fasta --max-kmer-len 99
```

`--prev-kmer-len K`

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 options for adjusting the quality, 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.

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.

`--max-kmer-store INT`

The maximum size per process for the k-mer store, in MB. This defaults to 1% of available memory. Higher values use more memory, but with potentially faster computation.

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

The maximum number of remote procedure calls (RPCs) outstanding at a time. This reduces memory usage but increases running time. It defaults to 100, and is interpreted as unlimited if this is set to 0.

`--use-heavy-hitters BOOL`

Activate code for managing *heavy hitters*, which are k-mers that occur far more frequently than any others. This can improve performance for datasets that have a few k-mers with extremely high abundance. Defaults to false.

`--ranks-per-gpu INT`

When GPUs are used, this overrides the automatic detection of how many processes use each GPU. It can be used to explicitly to tune for specific configurations of CPU/GPUs. This defaults to 0 if there are no GPUs available; otherwise it is automatically calculated.

`--force-bloom BOOL`

Always use bloom filters when analyzing k-mers. Bloom filters reduce memory usage and MHM2 determines if they are necessary by computing the available memory before doing the k-mer analysis. However, it is possible to force MHM2 to always use as little memory as possible by setting this option to true. It defaults to false.

`--pin STRING`

Restrict the hardware contexts that processes can run on. There are 4 options: `cpu`, meaning restrict each process to a single logical CPU; `core`, meaning restrict each process to a core; `numa`, meaning restrict each process to a NUMA domain; and `none`, meaning don't restrict the processes. By default, it restricts to `cpu`.

`--shared-heap INT`

Set the shared heap size used by the UPC++ runtime, as a percentage of the available memory. This is automatically set and should never need to be adjusted. However, there may be some rare cases on certain hardware where this needs to be adjusted to enable MHM2 to run without memory errors. This defaults to 10%.

`--procs INT`

Set the number of processes for MHM2 to run with. By default, MHM2 automatically detects the number of available processes and runs on all of them. 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.