Hermes Documentation

Evan Senter

June 24, 2014

Contents

1 Introduction

This documentation aims to outline the basic dependencies, installation, and usage of the Hermes RNA software suite, as well as provide extended commentary on the flags and options available to code. Particular attention is payed to the invocation style of mashup programs (using `multi_param`) to dispatch flags to the appropriate submodules. To get started as quickly as possible, see the quickstart one-liner in 2.1. Should you still have questions, you can reach the main author of the source code at evansenter@gmail.com.

2 Installation

2.1 Quick Start

From the root directory of Hermes, execute the following command:

```
cd build && cmake .. && make
```

If you encounter errors in configuring or compiling the software, we recommend checking out section 2.4 on common troubleshooting solutions.

2.2 Dependencies

`cmake` (≥ 2.6-patch 4, tested through 3.0.0) `http://www.cmake.org/`

CMake is used as the build system for Hermes.

GNU99 compiler:

Most C compilers should support the `-std=gnu99` flag, which is required for GNU library extensions, particularly `unistd.h`

C++11 compiler:

C++11 support (included in `g++` ≥ 4.7) is necessary for proper struct initialization in `FFTbor2D`.

OpenMP support `http://openmp.org/wp/`

OpenMP support comes by default in most modern compilers, and is required for loop-optimization in `FFTbor2D`.

`LAPACK` (≥ 3.4.2) `http://www.netlib.org/lapack/`

Various LAPACK coutines are used in `RNAmfpt`to compute the inverse, or pseudoinverse, of a transition probability matrix.

`GSL` (≥ 1.15, tested through 1.16) `http://www.gnu.org/software/gsl/`

GSL is required to compute the eigendecomposition of a (possibly) non-symmetric transition rate matrix for `RNAeq`.

`FFTW3` (≥ 3.3.4) `http://www.fftw.org/`

FFTW3 functions are used to compute the inverse discrete Fourier transform in `FFTbor2D`.

`libRNA.a` (≥ 2.0.7, tested through 2.1.7) `http://www.tbi.univie.ac.at/RNA/`

Various ViennaRNA functions and data structures are leveraged for homogenous energy model support, as well as `fold_par`, `pf_fold_par`, and `subopt_par`. We additionally make use of ViennaRNA functions to compute the maximum base pair distance between two structures, required for bounding the polynomial size in `FFTbor2D`.

2.3 Compilation

`cd build && cmake ..`

First, ensure that your system has the dependencies outlined above (the presence of CMake can be verified with `cmake --version`). While not required, is is widely considered *best practice* to perform an out-of-source build, where the compilation of the code happens in a separate directory from the location of the source itself. To this end, `hermes`provides an empty `build` directory that can be used for compiling the code. From `hermes/build` execute `cmake` with the path to `hermes/CMakeLists.txt` (`..`) provided as argument.

`make`

Compiles the code, and generates binary executables for `FFTbor2D`, `RNAmfpt`, `RNAeq`, `FFTmfpt`, `FFTeq`, and `RateEq`. Additionally generates both static and shared libraries for `FFTbor2D`, `RNAmfpt`, and `RNAeq`. The output directory for binaries is `hermes/bin` and output directory for libraries is `hermes/lib`.

`make install` (optional)

Installs the executables built with `make` to `$DESTDIR/bin` and copies libraries / archives to `$DESTDIR/lib` (on *nix systems, `$DESTDIR` defaults to `/usr/local`).

2.4 Troubleshooting

While we have done the utmost to try and ensure that CMake is able to infer locations of third-party libraries and add compiler-appropriate flags in an automated fashion, due to the diversity of build environments possible, it is possible that you will need to specify additional command-line flags to `cmake` when generating the Makefiles in order to successfully build Hermes. The following are five useful flags for CMake, and a brief explanation of when they may need to be employed:

- The default compiler I'd like to use for C code is installed in a non-standard location, or not the globally default C compiler.

  CMAKE_C_COMPILER

  i.e. `cmake -DCMAKE_C_COMPILER=/path/to/c/compiler ..`

  This variable sets the path to the compiler to use for configuration and subsequent compiliation via `make`. This is the compiler that will be used by CMake to test for the presence of various flags, i.e. `-O3` and `-Wall`.

- The default compiler I'd like to use for C++ code is installed in a non-standard location, or not the globally default C++ compiler.

  CMAKE_CXX_COMPILER

  i.e. `cmake -DCMAKE_CXX_COMPILER=/path/to/cxx/compiler ..`

  Same as above.

- Libraries required by Hermes are not installed in a location visible by the linker (in LD_LIBRARY_PATH), and CMake is unable to validate their existence.

  CMAKE_LIBRARY_PATH

  i.e. `cmake -DCMAKE_LIBRARY_PATH="/more/libraries;/even/more/libraries" ..`

  In the case when libraries required by Hermes are not visible to CMake, or the global library is an out-of-date version, it is possible to provide hints to the build tool for additional directories to search. Directories specified by the CMAKE_LIBRARY_PATH flag will be prepended onto the linker search path, and thus override global matches (handling the case where default libraries aren't sufficiently up to date). When desiring to provide multiple locations to search for libraries, CMake uses the semicolon (;) character as a separator and the entire string should be quoted to escape the shell environment.

- Headers required by Hermes are not installed in a location visible by the compiler (in CPATH or a derivative), and as a result I'm seeing `undefined reference to` errors.

  CMAKE_INCLUDE_PATH

  i.e. `cmake -DCMAKE_INCLUDE_PATH="/more/includes;/even/more/includes" ..`

  It is generally likely that the CMAKE_LIBRARY_PATH and CMAKE_INCLUDE_PATH will both be necessary, when either one is required. This flag operates in a fashion identical to CMAKE_LIBRARY_PATH described above, and uses the same syntax. Alternatively a user can update their `CPATH` environment variable, but this may have unpredictable results when headers are found, but out of date.

- I don't have permissions to `make install` to the default location (generally `/usr/local` for *nix) on my system.

  CMAKE_INSTALL_PREFIX

  i.e. `cmake -DCMAKE_INSTALL_PREFIX=/make/install/path/prefix ..`

  This variable sets the destination directory of the `make install` command. Binaries will be placed in the `bin` subdirectory and libraries will be placed in the `lib` subdirectory. This is analogous to `./configure --prefix=/make/install/path/prefix` in Autotools and can also be achieved by setting the `DESTDIR` environment variable.

3 Software Organization

3.1 General Principles

The Hermes code is organized into two major directories, `hermes/src` and `hermes/mashup`. The conceptual difference between these two directories is that `hermes/src` code (`FFTbor2D`, RNAmfpt, RNAeq) are all stand-alone pieces of software which achieve specific goals (computing energy grids, hitting time, and equilibrium time respectively). Alternatively, code residing in hermes/mashup aims to leverage general concepts across the Hermes package to provide a means to ask even more specific questions. In example, FFTbor2Dallows an investigator to compute the 2D energy grid correspondent to an input RNA sequence s and two structures A, B. RNAeqcan estimate the population occupancy of A, B for s by either sampling suboptimal structures or exhaustive structural enumeration, but these approaches aren't tractable for non-trivial RNAs.

FFTeq, located in hermes/mashup/population_from_fftbor2d uses functions from libfftbor_static.a (derived from FFTbor2D) to compute the energy landscape and functions from librnaeq_static.a (derived from RNAeq) to estimate population occupancy for s, A, B without requiring an investigator to copy pieces of individual packages to achieve their goals, instead using the static libraries automatically produced for all hermes/src software, shared headers made available in hermes/h and libmulti_param.a (from hermes/src/multi_param) to dispatch command-line arguments to the appropriate underlying function. The result? The entirety of FFTeqis 67 lines of C++ code, and uses native binary data structures the entire way through; there is no command-line funneling of FFTbor2Dinto RNAeq.

3.2 multi_param Overview

All of FFTbor2D, RNAmfpt, and RNAeqpresent a wide selection of command-line arguments to meet the diverse demands of end users. When we moved on to developing *mashup* software between these three programs, there were a number of requirements that we came up with to make both implementing and using these programs as easy as possible. We decided that *a*) the developer should not have to reimplement command-line parsing for mashups *b*) all existing flags for underlying libraries (i.e. FFTbor2D, RNAmfpt, RNAeq) would be supported, and *c*) flags would be namespaced to have deterministic targets.

To achieve these goals, the library multi_paramwas developed. This library simply takes a collection of command line arguments and re-dispatches them to the appropriate underlying library. Given a set of command-line flags such as `--all-v --fftbor2d-i GGGAAACCC --fftbor2d-j '.........' --fftbor2d-k '(((...)))' --mfpt-x --mfpt-h`, the code ensures that FFTbor2Dis passed `-v -i GGGAAACCC -j '.........' -k '(((...)))'` as options in argv and RNAmfptis passed `-v -x -h` as options in argv.

3.3 multi_param Details

In example from hermes/mashup/mfpt_from_fftbor2d/mfpt_from_fftbor2d.cpp:

```
PARAM_CONTAINER* params;
FFTBOR2D_PARAMS fftbor2d_params;

/* ...omitted for clarity... */

char* subparams[] = { "fftbor2d", "mfpt" };
params            = split_args(argc, argv, subparams, 2);

fftbor2d_params = init_fftbor2d_params();
parse_fftbor2d_args(fftbor2d_params, params[0].argc, params[0].argv);
```

The way this code works is by first declaring which packages can be used by the mashups, out of `fftbor2d`, `mfpt`, or `population`. In the snippet above the selection is saved in the variable `subparams`. All of the `FFTbor2D`, `RNAmfpt`, and `RNAeq`libraries have `init_*_params` functions available, which return an object with the default parameters for that package. They also all have `parse_*_args` functions, which take three arguments, *1*) a pointer to the parameters object *2*) `argc`, and *3*) `argv` .

`split_args` takes the prefixed command-line arguments (see the example in 3.2) and bins them by their prefix, with the special `--all` prefix being supplied to all declared subparams. The prefixes are then removed and the grouped arguments are returned from the function in a `PARAM_CONTAINER` array, with the same order as the subparams were passed to the function. It is then trivial to call the `parse_*_args` functions with the corresponding subarrays from `PARAM_CONTAINER` to get a final parameters object.

Leveraging the example from 3.2 a final time, the variable `params` would look as follows after invoking `split_args`:

```
[
  { argv: ["-v", "-i", "GGGAAACCC", "-j", ".........", "-k", "(((...)))"], argc: 7 },
  { argv: ["-v", "-x", "-h"], argc: 3 }
]
```

4 Core Programs

4.1 FFTbor2D

4.2 RNAmfpt

4.3 RNAeq

5 Mashups

5.1 FFTmfpt

5.2 FFTeq

5.3 RateEq