New Lattice Microbes v2.3 Installation Guide - Additional Instructions for Dependencies

Benjamin R. Gilbert

April 5, 2020

This guide in combination with the `README` in the HeLa model repository should be used to build the Lattice Microbes HeLa model detailed in

Zhaleh Ghaemi, Joseph R. Peterson, Martin Gruebele, and Zaida Luthey-Schulten. An in-silico human cell model reveals the influence of spatial organization on rna splicing. *PLOS Computational Biology*, 16(3):1–23, 03 2020

1 Software Requirements

All of the dependencies that must be installed from source and the related bash functions are found in the `LM_2.3_dependency_sources.tar`. Extracting this archive should result in a directory named `Software` and a text file named `bashrc_functions_LM_2.3.txt`.

```
# cd /home/"user"/Downloads
# tar -xvf LM_2.3_dependency_sources.tar
```

The directory contains the source code for the dependencies that must be installed from source; the subdirectories within it follow the layout specified in the directions for the build and source directory locations and the whole directory may be copied to the user's home directory.

```
# cp -r /home/"user"/Downloads/Software /home/"user"/Software
```

Completing this step allows the user to avoid downloading the sources individually and to skip the steps titled "Create the directory for the builds and the source code" and "Place the tar file in the build and source directory," which are found within each dependency's instructions for installing from source.

The text file contains bash functions used to create a shell environment suitable for building and running Lattice Microbes v2.3.

1.1 List of All Dependencies

- Lattice Microbes v2.3 — Download compressed tar file
- GCC v8.4.0 — Install from source
- CUDA v10.1 — Download and execute runfile
- HDF5 v1.12.0 — Install from source
- Protocol Buffers v3.11.4 — Install from source
- PCRE v8.44 — Install from source, needed for SWIG
- Boost v1.72.0 — Install from source, needed for SWIG
- SWIG v4.0.1 — Install from source using PCRE and Boost libraries
- Python 2.7 — Create new conda environment
- NumPy and SciPy — Install in conda environment
- iGraph — Install in conda environment
- pygexf — Install in conda environment
- lxml — Install in conda environment
- matplotlib — Install in conda environment
- SBML v5.18.0 — Download tar file with libraries

1.2 Conda Environment - Includes Python and Python Packages

This guide assumes the Anaconda Python distribution has already been installed on the machine, if this is not the case, install Anaconda before proceeding.

The new conda environment will be named **LM_2.3_base** and will be created with Python2.7. Packages within the virtual environment will be handled by the conda package manager, it should automatically satisfy the various dependencies.

1. Create new conda environment.
   ```
   # conda create --name LM_2.3_base python=2.7
   ```
   If you want to use Jupyter notebooks with the LM install, instead use the command.
   ```
   # conda create --name LM_2.3_base anaconda python=2.7
   ```
2. Add the packages to the virtual environment and allow it to solve the dependencies.
   ```
   # conda install -c auto pygexf
   # conda install numpy scipy h5py lxml matplotlib
   ```

1.3 GCC v8.4.0

Install location:`/usr/local/Compilers/GCC/8.4.0`

1. Create the install directory.
   ```
   # cd /usr/local
   # sudo mkdir -p ./Compilers/GCC/8.4.0
   ```
2. Create the directory for the builds and the source code.
   ```
   # mkdir -p /home/"user"/Software/Compilers/GCC/8.4.0
   ```
3. Place the tar file in the build and source directory.
   ```
   # cd /home/"user"/Software/Compilers/GCC/8.4.0
   # mv gcc-8.4.0.tar.gz .
   ```
4. Extract the source code
   ```
   # tar -xzvf gcc-8.4.0.tar.gz
   ```
5. Test if pre-requisites are present.
   ```
   # cd gcc-8.4.0
   # ./contrib/download_prerequisites
   ```
6. Create the build directory.
   ```
   # cd ..
   # mkdir gcc-build
   # cd gcc-build
   ```
7. Configure GCC
   ```
   # ../gcc-8.4.0/configure -v\
     --build=x86_64-linux-gnu --host=x86_64-linux-gnu --target=x86_64-linux-gnu\
     --prefix=/usr/local/Compilers/GCC/8.4.0\
     --enable-checking=release --enable-languages=c,c++,fortran --disable-multilib\
     --program-suffix=-8.4.0
   ```
8. Build GCC. (8 is the number of cores to be used, change accordingly.)
   `# make -j 8` This will take a while and it is suggested that the user jump ahead to 2 and 3.
9. Install GCC.
   ```
   # sudo make install
   ```

Versions 8.4.0 of all of the compilers in the Gnu Compiler Collection and the related libraries are now installed at `/usr/local/Compilers/GCC/8.4.0`. Explicitly using these, rather than those installed via some form of package manager, will be accomplish through use of functions added to the user's `.bashrc`. Skip ahead to 3 and complete that step before using the new compiler to build the remaining dependencies. Remember to source your `.bashrc` before attempting to use the new functions.

After completing this step the new compiler can be tested by loading it and then checking the version number.

```
# GNU_NEW_LOAD 8.4.0
# gcc --version
```

1.4 CUDA v10.1

Install CUDA v10.1 using the runfile and instructions provided at the Nvidia website.

Install location:`/usr/local/cuda-10.1`

After installing CUDA v10.1 edit the local.mk to match the compute capabilities of the installed Nvidia GPU.

1.5 HDF5 v1.12.0

Build and install HDF5 v1.12.0 using the install of gcc-8.4.0.

Install location:`/usr/local/Libraries/hdf5/1.12.0_gcc8.4.0`

1. Confirm that gcc-8.4.0 is loaded
   ```
   # GNU_NEW_LOAD 8.4.0
   # gcc --version
   ```
   The output should show that v8.4.0 is being used.
2. Create the install directory.
   ```
   # cd /usr/local
   # sudo mkdir -p ./Libraries/hdf5/1.12.0_gcc8.4.0
   ```
3. Create the directory for the builds and the source code.
   ```
   # mkdir -p /home/"user"/Software/Libraries/hdf5/1.12.0
   ```
4. Place the tar file in the build and source directory.
   ```
   # cd /home/"user"/Software/Libraries/hdf5/1.12.0
   # mv hdf5-1.12.0.tar.gz .
   ```
5. Extract the source code
   ```
   # tar -xzvf hdf5-1.12.0.tar.gz
   ```
6. Create the build directory.
   ```
   # cd ..
   # mkdir build_gcc8.4.0
   # cd build_gcc8.4.0
   ```

7. Configure the hdf5 build.
   ```
   # ../hdf5-1.12.0/configure --prefix=/usr/local/Libraries/hdf5/1.12.0_gcc8.4.0\
     --enable-cxx
   ```

8. Build hdf5. (8 is the number of cores to be used, change accordingly.)
   ```
   # make -j 8
   ```

9. Install hdf5.
   ```
   # sudo make install
   ```

The HDF5 library caused some additional difficulties when running LM and the solution was to create a symbolic link.

```
# ln -s /usr/local/Libraries/hdf5/1.12.0_gcc8.4.0/lib/libhdf5.so.200.0.0\
  /lib/x86_64-linux-gnu/libhdf5.so.200
```

1.6 Protocol Buffers v3.11.4

Build and install Protocol Buffers v3.11.4 using the install of gcc-8.4.0.

Install location:`/usr/local/Libraries/protobuf/3.11.4_gcc8.4.0`

1. Confirm that gcc-8.4.0 is loaded
   ```
   # GNU_NEW_LOAD 8.4.0
   # gcc --version
   ```
   The output should show that v8.4.0 is being used.

2. Create the install directory.
   ```
   # cd /usr/local
   # sudo mkdir -p ./Libraries/protobuf/3.11.4_gcc8.4.0
   ```

3. Create the directory for the builds and the source code.
   ```
   # mkdir -p /home/"user"/Software/Libraries/protobuf/3.11.4
   ```

4. Place the tar file in the build and source directory.
   ```
   # cd /home/"user"/Software/Libraries/protobuf/3.11.4
   # mv protobuf-all-3.11.4.tar.gz .
   ```

5. Extract the source code
   ```
   # tar -xzvf protobuf-all-3.11.4.tar.gz
   ```

6. Create the build directory.
   ```
   # mkdir build_gcc8.4.0
   # cd build_gcc8.4.0
   ```

7. Configure the protobuf build.
   ```
   # ../protobuf-3.11.4/configure --prefix=/usr/local/Libraries/protobuf/3.11.4_gcc8.4.0\
     "CFLAGS=-fPIC" "CXXFLAGS=-fPIC"
   ```

8. Build protobuf. (8 is the number of cores to be used, change accordingly.)
   ```
   # make -j 8
   ```
9. Install protobuf.
   ```
   # sudo make install
   ```

1.7 PCRE v8.44

Build and install PCRE v8.44 using the install of gcc-8.4.0.

Install location:`/usr/local/Libraries/PCRE/8.44_gcc8.4.0`

1. Confirm that gcc-8.4.0 is loaded
   ```
   # GNU_NEW_LOAD 8.4.0
   # gcc --version
   ```
   The output should show that v8.4.0 is being used.
2. Create the install directory.
   ```
   # cd /usr/local
   # sudo mkdir -p ./Libraries/PCRE/8.44_gcc8.4.0
   ```
3. Create the directory for the builds and the source code.
   ```
   # mkdir -p /home/"user"/Software/Libraries/PCRE/8.44
   ```
4. Place the tar file in the build and source directory.
   ```
   # cd /home/"user"/Software/Libraries/PCRE/8.44
   # mv pcre-8.44.tar.bz2 .
   ```
5. Decompress the source code
   ```
   # bzip2 -d pcre-8.44.tar.bz2
   ```
6. Extract the source code
   ```
   # tar -xvf source.tar
   ```
7. Create the build directory.
   ```
   # mkdir build_gcc8.4.0
   # cd build_gcc8.4.0
   ```
8. Configure the PCRE build.
   ```
   # ../pcre-8.44/configure --prefix=/usr/local/Libraries/PCRE/8.44_gcc8.4.0\
     --enable-unicode-properties\
     --enable-pcre16 --enable-pcre32
   ```
9. Build PCRE. (8 is the number of cores to be used, change accordingly.)
   ```
   # make -j 8
   ```
10. Install PCRE.
    ```
    # sudo make install
    ```

The PCRE library caused some additional difficulties when building and the solution was to create a symbolic link.

```
# ln -s /usr/local/Libraries/PCRE/8.44_gcc8.4.0/lib/libpcre.so.1.2.12\
  /lib/x86_64-linux-gnu/libpcre.so.1
```

1.8 boost v1.72.0

Build and install boost v1.72.0 using the install of gcc-8.4.0.

Install location:`/usr/local/Libraries/boost/1.72.0_gcc8.4.0`

1. Confirm that the conda environment with Python2.7 is active.
   ```
   # conda activate LM_2.3_base
   ```
2. Confirm that gcc-8.4.0 is loaded
   ```
   # GNU_NEW_LOAD 8.4.0
   # gcc --version
   ```
   The output should show that v8.4.0 is being used.
3. Create the install directory.
   ```
   # cd /usr/local
   # sudo mkdir -p ./Libraries/boost/1.72.0_gcc8.4.0
   ```
4. Create the directory for the builds and the source code.
   ```
   # mkdir -p /home/"user"/Software/Libraries/boost/1.72.0
   ```
5. Place the tar file in the build and source directory.
   ```
   # cd /home/"user"/Software/Libraries/PCRE/8.44
   # mv boost_1_72_0.tar.gz .
   ```
6. Extract the source code and move to source directory.
   ```
   # tar -xzvf boost_1_72_0.tar.gz
   # cd boost_1_72_0
   ```
7. Configure the boost build.
   ```
   # ./bootstrap.sh --prefix=/usr/local/Libraries/boost/1.72.0_gcc8.4.0
   ```
8. Stage the boost build. (8 is the number of cores to be used, change accordingly)
   ```
   # ./b2 stage -j8 threading=multi link=shared
   ```
9. Install boost
   ```
   # sudo ./b2 install threading=multi link=shared
   ```

1.9 SWIG v4.0.1

Build and install SWIG v4.0.1 using the install of gcc-8.4.0 and linking to the PCRE and boost libraries.

Install location: `/usr/local/Tools/swig/4.0.1_gcc8.4.0`

1. Confirm that the conda environment with Python2.7 is active.
   ```
   # conda activate LM_2.3_base
   ```
2. Confirm that gcc-8.4.0 is loaded
   ```
   # GNU_NEW_LOAD 8.4.0
   # gcc --version
   ```
   The output should show that v8.4.0 is being used.
3. Create the install directory.
   ```
   # cd /usr/local
   # sudo mkdir -p ./Tools/swig/4.0.1_gcc8.4.0
   ```
4. Create the directory for the builds and the source code.
   ```
   # mkdir -p /home/"user"/Software/Swig
   ```
5. Place the tar file in the build and source directory.
   ```
   # cd /home/"user"/Software/Tools/swig/4.0.1
   # mv swig-4.0.1.tar.gz .
   ```
6. Extract the source code
   ```
   # tar -xzvf swig-4.0.1.tar.gz
   ```
7. Create the build directory.
   ```
   # mkdir build_gcc8.4.0
   # cd build_gcc8.4.0
   ```
8. Configure the swig build.
   ```
   # ../swig-4.0.1/configure --prefix=/usr/local/Tools/swig/4.0.1_gcc8.4.0\
     --with-pcre-prefix=/usr/local/Libraries/PCRE/8.44_gcc8.4.0\
     --with-boost=/usr/local/Libraries/boost/1.72.0_gcc8.4.0\
     "PCRE_CONFIG=/usr/local/Libraries/PCRE/8.44_gcc8.4.0/bin/pcre-config"
   ```
9. Build swig(8 is the number of cores to be used, change accordingly.)
   ```
   # make -j 8
   ```
10. Test swig.
    ```
    # make -k check
    ```
11. Install swig.
    ```
    # sudo make install
    ```

1.10 SBML v5.18.0

Install SBML v5.18.0 using the pre-compiled Ubuntu binaries. Install location:`/usr/local/Libraries/SBML/5.18.0`

1. Create the install directory.
   ```
   # cd /usr/local
   # sudo mkdir -p ./Libraries/SBML/5.18.0
   ```
2. Create the directory for the builds and the source code.
   ```
   # mkdir -p /home/"user"/Software/Libraries/SBML/5.18.0
   ```
3. Place the tar file in the build and source directory.
   ```
   # cd /home/"user"/Software/Libraries/SBML/5.18.0
   # mv libSBML-5.18.0-Linux-x64-binaries-ubuntu.tar.gz .
   ```
4. Extract the binary files.
   ```
   # tar -xzvf libSBML-5.18.0-Linux-x64-binaries-ubuntu.tar.gz
   ```
5. Navigate the directory of binaries.
   ```
   # cd libSBML-5.18.0-linux
   ```
6. To avoid possible accidents, it is strongly suggested that you carefully change the name of the subdirectory `usr` without using any form of root permissions.
   ```
   # mv ./usr temp_dir
   # cd temp_dir
   ```
7. Copy the files within `temp_dir` to the install location.
   ```
   # sudo cp -r * /usr/local/Libraries/SBML/5.18.0/.
   ```

2 Configuring local.mk and editing Makefile/subdir.mk(s)

2.1 Configuring `local.mk` to use installed software.

Copy the local.mk.linux from to the main directory of the Lattice Microbes source code, the location of this directory depends on the user.

```
# cd LM_source_dir
# cp ./docs/config/local.mk.linux local.mk
```

Begin editing the local.mk using the text editor of your choice to specify build options and the locations and configurations of dependencies.

Proceeding from top to bottom of an unmodified local.mk (line numbers assume the preceeding changes have been made):

1. On line 29 change the `INSTALL_PREFIX` variable to the desired installation directory. Make sure that this is the same as the bash variable `LM_DIR` in the bash function `LM_2.3_LOAD`.

   ```
   INSTALL_PREFIX := /home/"user"/Workspace/LM/LM_2.3
   ```

2. On line 40 change the `HDF5_DIR` variable to match the location of the installed hdf5 library.

   ```
   HDF5_DIR := /usr/local/Libraries/hdf5/1.12.0_gcc8.4.0
   ```

3. On line 52 change the `PROTOBUF_DIR` variable to match the location of the installed protobuf library.

   ```
   HDF5_DIR := /usr/local/Libraries/hdf5/1.12.0_gcc8.4.0
   ```

4. On line 67 change the `CUDA_DIR` variable to match the location of the CUDA installation.

   ```
   CUDA_DIR := /usr/local/cuda-10.1
   ```

5. On lines 70-73 comment all of them out and add a line below on line 74 to specify the CUDA architecture being used in the `CUDA_ARCH` variable.

   ```
   CUDA_ARC := -gencode arch=compute_XX,code=compute_XX
   ```

6. On line 110 change the `PYTHON_SWIG` variable to match the location of the swig installation's binary.

   ```
   PYTHON_SWIG := /usr/local/Tools/swig/4.0.1_gcc8.4.0/bin/swig
   ```

7. On lines 114 and 115 comment both of them out and add lines below line 115 to specify variables used for building the Python wrapper. The last line should be line 122. These variables should reference where the user's conda virtual environments are installed. Depending on the user's Anaconda installation, these may be different, what is shown below is an example.

   ```
   PYTHON_INCLUDE_DIR := $(shell python-config --includes)
   PYTHON_LIB := $(shell python-config --libs)
   PYTHON_INCLUDE_DIR += -I/home/"user"/.conda/envs/LM_2.3_base/include/python2.7
   PYTHON_INCLUDE_DIR += -I/home/"user"/.conda/envs/LM_2.3_base/lib/\
   python2.7/site-packages/numpy/core/include
   PYTHON_LIB += -L/home/"user"/.conda/envs/LM_2.3_base/lib
   PYTHON_CONFIG_INFO_cflags := $(python-config --cflags)
   PYTHON_CONFIG_INFO_ldflags := $(python-config --ldflags)
   ```

8. On line 136 change the `SBML_DIR` variable to match the location of the installed SBML library.

```
SBML_DIR := /usr/local/Libraries/SBML/5.18.0
```

9. Below line 161 add the following.
```
CCFLAGS += $(PYTHON_CONFIG_INFO_cflags)
```

10. Below line 169 add the following.
```
LDFLAGS += $(PYTHON_CONFIG_INFO_ldflags)
```

11. Comment line 186 and uncomment line 187.
12. On line 191 change the `CUDA_INCLUDE_DIR` variable to the following.
```
CUDA_INCLUDE_DIR := -I$(CUDA_DIR)/targets/x86_64-linux/include
```

13. On line 192 change the `CUDA_LIB_DIR` variable to the following.
```
CUDA_LIB_DIR := -L$(CUDA_DIR)/targets/x86_64-linux/lib
```

14. On line 193 change the `CUDA_LIB` variable to the following.
```
CUDA_LIB := -lcudart $(CUDA_DIR)/targets/x86_64-linux/lib/libcudart_static.a
```

2.2 Makefile edits

Comment out line 158 and place the following underneath it.
```
all:  protobuf $(MAIN) util vmdplugin pylm pylmExamples pylmInstall
```

2.3 subdir.mk edits

3 Setting Up Environment

3.1 Additions to `.bashrc`

Make these additions to your `.bashrc` using the text editor of your choice. Remember to source your `.bashrc` after completing the additions. All of the changes to path variables will only be used when the functions are called in the shell, so it is safe to source your `.bashrc` without worrying about unexpected consequences.
```
# source /home/"user"/.bashrc
```

3.1.1 Compiler Loading

```
#GNU-Compilers
   function GNU_LOAD {
    #Set aliases for use in cmake execution
    alias gcc=$GCC_INSTALL/'gcc-'$1
    alias cc=$GCC_INSTALL/'gcc-'$1
    alias g++=$GCC_INSTALL/'g++-'$1
    alias c++=$GCC_INSTALL/'c++-'$1
    alias gfortran=$GCC_INSTALL/'gfortran-'$1
    #export environment variables for use in cmake execution
    export FORTRAN_COMPILER=$GCC_INSTALL/'gfortran-'$1
    export FC=$GCC_INSTALL/'gfortran-'$1
    export F77=$GCC_INSTALL/'gfortran-'$1
    export F90=$GCC_INSTALL/'gfortran-'$1
    export CMAKE_Fortran_COMPILER=$GCC_INSTALL/'gfortran-'$1
    export CMAKE_C_COMPILER=$GCC_INSTALL/'gcc-'$1
    export CC=$GCC_INSTALL/'gcc-'$1
    export CXX=$GCC_INSTALL/'g++-'$1
   }
   export -f GNU_LOAD

   function GNU_NEW_LOAD {
    #export environment variables for GNU compiler
    export temp_ver=$1
    export GCC_DIR='/usr/local/Compilers/GCC/'$temp_ver
    export GCC_INC=$GCC_DIR/'include'
    export GCC_LIB=$GCC_DIR/'lib64'
    export LD_LIBRARY_PATH=$GCC_LIB:$LD_LIBRARY_PATH
    export GCC_INSTALL=$GCC_DIR/'bin'
    GNU_LOAD $temp_ver
   }
   export -f GNU_NEW_LOAD
```

3.1.2 CUDA Loading

```
function CUDA_NEW_LOAD_10.1 {
    export PATH=/usr/local/cuda-10.1/bin:/usr/local/cuda-10.1/\
       nsightCompute-2019.4.0${PATH:+:${PATH}}
    export LD_LIBRARY_PATH=/usr/local/cuda-10.1/\
       lib64${LD_LIBRARY_PATH:+:${LD_LIBRARY_PATH}}
   }
   export -f CUDA_NEW_LOAD_10.1
```

3.1.3 Library Loading

```
#PCRE
   function PCRE_NEW_LOAD {
    PCRE_DIR=’/usr/local/Libraries/PCRE/8.44_gcc8.4.0’
    PCRE_LIB=$PCRE_DIR/’lib’
    export LD_LIBRARY_PATH=$PCRE_LIB:$LD_LIBRARY_PATH
   }
   export -f PCRE_NEW_LOAD


   #hdf5
   function hdf5_NEW_LOAD {
    hdf5_DIR=’/usr/local/Libraries/hdf5/1.12.0_gcc8.4.0’
    hdf5_LIB=$hdf5_DIR/’lib’
    export LD_LIBRARY_PATH=$hdf5_LIB:$LD_LIBRARY_PATH
   }
   export -f hdf5_NEW_LOAD


   #protobuf
   function protobuf_NEW_LOAD {
    protobuf_DIR=’/usr/local/Libraries/protobuf/3.11.4_gcc8.4.0’
    protobuf_LIB=$protobuf_DIR/’lib’
    export LD_LIBRARY_PATH=$protobuf_LIB:$LD_LIBRARY_PATH
   }
   export -f protobuf_NEW_LOAD


   #SBML
   function SBML_NEW_LOAD {
    SBML_DIR=’/usr/local/Libraries/SBML/5.18.0’
    SBML_LIB=$SBML_DIR/’lib64’
    export LD_LIBRARY_PATH=$SBML_LIB:$LD_LIBRARY_PATH
   }
   export -f SBML_NEW_LOAD
```

3.1.4 Lattice Microbes Path Variables

```
function LM_2.3_LOAD {
    PCRE_NEW_LOAD
    hdf5_NEW_LOAD
    protobuf_NEW_LOAD
    SBML_NEW_LOAD
    LM_DIR=’/home/"user"/Workspace/LM/LM_2.3’
```

```
 export PATH=$PATH:$LM_DIR/'bin'
 export PYTHONPATH=$LM_DIR/'lib/lm':$PYTHONPATH
 export PYTHONPATH=$LM_DIR/'lib/python':$PYTHONPATH
}
export -f LM_2.3_LOAD
```

4 Loading Environment Variables Prior to Build

Before building and executing Lattice Microbes v2.3 the environment variables must be set using the conda environment and the bash functions added to the user's `.bashrc`.

1. Activate the conda environment.
   `# conda activate LM_2.3_base`
2. Load the CUDA compiler and libraries.
   `# CUDA_NEW_LOAD_10.1`
3. Load the gcc compiler.
   `# GNU_NEW_LOAD 8.4.0`
4. Load the libraries and add install location to the Python path.
   `# LM_2.3_LOAD`

5 Building and Installing Lattice Microbes

If the user wants to visualize the results of Lattice Microbes simulations, they should build and install Lattice Microbes with VMD.

5.1 Without VMD

Install Lattice Microbes

1. Navigate to main directory of the Lattice Microbes source code.
   `# cd LM_source_dir`
2. Make Lattice Microbes using the settings in `local.mk`. As a note, difficulties were encountered when using multiple threads for the build (build option `-j 8`).
   `# make LOCALMK=local.mk`
3. Install the build at the `INSTALL_PREFIX` specified in the local.mk.
   `# make install`

5.2 With VMD

Assuming VMD has been previously installed at `/usr/local/VMD/"VMD version number"` with the VMD binary in `/usr/local/VMD/"VMD version number"/bin` and the supporting libraries, binaries, and plugins in `/usr/local/VMD/"VMD version number"/vmd`, the `local.mk` will be edited to match these installation locations.

Configure local.mk

1. On line 148 enable VMD by changing the `USE_VMD` variable. `USE_VMD := 1`

2. On line 151 specify the location fo the VMD installation using the `VMD_DIR` variable. `VMD_DIR := /usr/local/VMD/"VMD version number"/vmd`

Install Lattice Microbes

1. Navigate to main directory of the Lattice Microbes source code.
   ```
   # cd LM_source_dir
   ```
2. Make Lattice Microbes using the settings in `local.mk`. As a note, difficulties were encountered when using multiple threads for the build (build option `-j 8`).
   ```
   # make LOCALMK=local.mk
   ```
3. Install the build at the `INSTALL_PREFIX` specified in the local.mk. Root permissions are required because the VMD plug-in is being installed at the location of the VMD installation, which is by default within the directory `/usr/local`.
   ```
   # sudo make install
   ```

References

[1] Zhaleh Ghaemi, Joseph R. Peterson, Martin Gruebele, and Zaida Luthey-Schulten. An in-silico human cell model reveals the influence of spatial organization on rna splicing. *PLOS Computational Biology*, 16(3):1–23, 03 2020.