Multidimensional Replica Exchange Umbrella Sampling with the LAMMPS Ensembles Interface
version 0.92

Adrian W. Lange

July 18, 2013

1 Introduction

The purpose of this work is to document how to use the Multidimensional Replica Exchange Umbrella Sampling (REUS) part of the LAMMPS Ensemble (LE) interface. The LE driver is an external executable program that links to a library build of LAMMPS. The LAMMPS library must be compiled with the USER-COLVARS and USER-RAPTOR packages installed. [1] We will assume the user has some familiarity with compiling and running LAMMPS already.

1.1 Note to users from the author

LE is not guaranteed to be bug-free. If you find a bug, take the initiative to fix it yourself because I, Adrian, won't be offering much (read: none) support for this code in the future. Also, feel free to experiment and add to the code. Science!

2 System Requirements

The LE is a parallel program written in C, and it should be compiled with a MPI C compiler (*e.g.*, mpicc). However, sometimes a strict C compiler might be finicky, so using a MPI C++ compiler (*e.g.*, mpicxx) will also work.

3 Installation

In the future, I would like there to be straightforward scripts to install this code, but until then, the user will just have to follow these instructions in sequence.

There are 3 major steps:

1. Obtain source code

2. Compile LAMMPS library

3. Compile LE driver

---

[1] But, a clever programmer could easily figure out how to decouple those packages if he/she wanted.

Each of these steps has further sub-steps involved, which we describe in detail below.

3.1 Obtain source code

At the moment, you'll need to obtain the LE code and the modified LAMMPS library code directly from me, Adrian. Feel free to email me about it at adrianwlange@gmail.com.

Alternatively, there is currently also a GitHub repository of the code at:

`https://github.com/awlange/lammps_ensembles`

LE has been successfully run with the LAMMPS version from March 9, 2013. For convenience, a tarball of this version is provided with the LE package. In the rest of this manual, we will use path names involving this version of LAMMPS, but you will need to change it appropriately for your set up. If you plan to use this tarball, unpack it with `tar -xzf lammps.tar.gz`.

LE *does not* ship with the USER-RAPTOR package (*i.e.*, the RAPTOR code). RAPTOR has not been released publicly yet, and so we don't want to post it on GitHub. Also, without that package, LE will not compile and will have limited use, if any at all. This all means that you will need to obtain the latest version of USER-RAPTOR from the SVN repository maintained by the Voth group.

3.2 Compile LAMMPS

Step 1: Set up your Makefile. You will need to `cd` into `lammps-9Mar13/src/MAKE` and edit your Makefile to make sure that it is set up for your system. There are some examples provided if you need to set up one from scratch. Also, you may find Makefiles for various machines on the Voth Wiki.

NOTE: In order to use the λ-Exchange capability of LE, you *must* compile with the macro `-DRELAMBDA` defined in your Makefile. LE may fail to compile without this macro definition.

You *should* compile with OpenMP compiler flags turned on, which tells the compiler to multi-thread loops and sections where OpenMP pragmas exist. How to specify the flag differs depending on which compiler you are using. For example, GCC compilers usually require the flag `-fopenmp` to be included, Intel compilers usually require the flag `-openmp`, and PG compilers usually require the flag `-mp`. Consult your compiler's manual for more details. Compiling with OpenMP turned on is actually not required, but it is highly recommended because it provides the user greater flexibility to make use of a given computer system. And your code will run faster.

Step 2: Install the USER-OMP package. Type `make yes-USER-OMP` in `lammps-9Mar13/src`. This installs the OpenMP threaded routines that will benefit from having the OpenMP flags turned on.

Step 3: Install the USER-COLVARS package. Type `make yes-USER-COLVARS` in `lammps-9Mar13/src`. This installs the interface to the colvars library provided with LAMMPS.

Step 4: Install the USER-RAPTOR package. Type `make yes-USER-RAPTOR` in `lammps-9Mar13/src`. This installs the RAPTOR code for MS-EVB simulations. Again, remember to define the macro `-DRELAMBDA` in your Makefile to compile the λ-Exchange code.

Optional step. Install any other packages you may need in LAMMPS here. Otherwise, move on to the next step.

Step 6: Add the LE files to LAMMPS source. There are a handful of files that need to be added to the LAMMPS source code in order for LE to work. They are contained in the directory `add_to_lammps`. Copy these over to the LAMMPS source directory, overwriting what is in there. (In the future, we will probably make this a USER package to simply.)

Step 7: Add the LE files to colvars library source. There are a handful of files that need to be added to the LAMMPS source code in order for LE to work. They are contained in the directory `add_to_colvars`. Copy these over to the clovers library source directory (`lammps-9Mar13/lib/colvars`), overwriting what is in there.

Step 8: Compile the colvars library. Change your directory to `lammps-9Mar13/lib/colvars`. You can modify the makefile Makefil.g++ if you like or copy it over to a new modified one if you need to for your system. It is compiled as a serial library. Once you are satisfied with the Makefile, type `make -f Makefile.g++` (or substitute your modified Makefile if you did so) to make the library.

Step 9: Compile LAMMPS as a library. This is documented on the LAMMPS web site, but it follows a simple procedure similar to the usual executable compilation. Change directories into the `lammps-9Mar13/src` directory and type

```
make makelib
make -f Makefile.lib foo
```

where foo is the machine name, corresponding to your Makefile. Then, wait a couple minutes (or longer if optimization is cranked up) for compilation. If all goes well, you will end up with a binary file named `liblmp_foo.a`. It is this file that you will be linking LE against.

3.3 Compile the LAMMPS Ensembles Driver

Step 1: Set up your Makefile. You will need to `cd` into `ens_src` and edit your Makefile to make sure that it is set up for your system, just like for the LAMMPS build. In the Makefile, you will need modify only a few variables manually for your system:

- `LAMMPSDIR = dir/foo/src`
  This is the path of your LAMMPS library build from Section 3.2. Set `dir/foo/src` to the appropriate path. Do not leave any spaces at the end of the line.

- `LIBDIR = foo/fftw/lib/`
  This specifies any external libraries that LAMMPS should be linked with. For example, if you compiled LAMMPS with FFTW, then you need to specify the path of the FFTW library here. Set `dir/foo/src_lammps_lib/` to the appropriate path.

- `LIB = -llmp_foo -L$(LIBCOLVARS) -lcolvars -lfftw3f`
  These are the binary filenames of the libraries that you are linking with, following the usual convention that a file like `liblmp_foo.a` is included as `-llmp_foo`, removing the "ib" and ".a". You will need to specify the LAMMPS library binary as well as any other libraries LAMMPS is to be linked with, such as a certain FFTW library. The example, `-lfftw3f`, would be needed if you wanted to link with FFTW3 with single precision. Be aware that the ordering from left to right of the libraries listed is important, as is usually the case in Makefiles.

- `DEBUG = -g`

  This is for optional debugging flags. It can be left blank. Including `-g` is useful for compiling with line references for stack traces or using the `gdb` debugger. You may also choose to specify the preprocessor macros `-DREUS_DEBUG`, `-DMREUS_DEBUG`, `-DRELAMBDA_DEBUG`, or `-DTEMPER_DEBUG` for extra printed output that shows what the random numbers and energies are being used for the Metroplis acceptance criteria during attempted exchanges. Other debug flags can be found in the source code. However, it is not recommended to keep these debug macros on when performing production runs, as it will significantly increase the size of your output file.

- `CCFLAGS = -O3 -fopenmp $(DEBUG) -I$(LAMMPSSRC)`

  This is for any additional compiler flags for the C compiler. The optimization level is not very important since not much time is actually spent doing computations in the REUS driver, as most of it takes place in LAMMPS. However, you should leave the `$(DEBUG) -I$(LAMMPSSRC)` part intact. You will need to include OpenMP flags also, where the above example is for gcc style compiler.

Step 2: Compile LE. As you can see from the Makefile, there are really only two commands you will need for making the REUS driver. To compile from scratch, simply type

```
make
```

If all goes well, you wll end up with an executable named **ens_driver**, which is short for "Ensembles Driver." That's it! You can now run LE.

The only other command you may need is for deleting your object files, which is done by typing (don't do this unless you want to re-compile, though)

```
make clean
```

4   Using LAMMPS Ensembles for Multidimensional REUS

The idea behind LE is to split the MPI universal communicator into separate subcommunicators, each of which launches its own LAMMPS run. Each subcommunicator corresponds to a replica in the REUS algorithm, and each subcommunicator reads a specified LAMMPS input file. This means we need to set up a seperate input file for each replica we want to run. For the REUS algorithm to be useful, each replica should have a different umbrella bias potential not too far from at least one other replica's bias potential. This is something the user will need to tinker with to find the best acceptance ratio. The umbrella sampling bias potential is controlled via the colvars library, for which there is extensive documentation online at the LAMMPS website.

We will discuss the different types of replica exchange possible with LE below. The types supported (i.e., ones that are stable) currently are:

- TEMPER: Parallel tempering (i.e., temperature replica exchange).

- REUS: Replica Exchange Umbrella Sampling for MS-EVB and RAPTOR.

- RELAMBDA: λ-Exchange (Hamiltonian exchange) for MS-EVB and RAPTOR, off-diagonal scaling.

- COLVARX: REUS using the colvars library. (Limited options right now). Not for MS-EVB.

- MREUS: Multi-dimensional REUS, TEMPER, and/or RELAMBDA.

Only MREUS is multi-dimensional. All others are one-dimensional.

The coordinate exchange code (COORDX) is experimental. If it were working more properly, it would be able to support arbitrary dimensions and many types of Hamiltonian exchange. Currently, it only seems to work when only a few (about less than 8) processors are assigned to each replica. The issue is that there is some information being lost during coordinate exchange at higher processor counts, and this issue is unresolved. If you want to fix it, go for it! Otherwise, please only use the stable replica exchange methods.

4.1 TEMPER

Parallel tempering, temperature replica exchange.

This routine *does not* require the user to be running MS-EVB. It can be used for any old parallel tempering calculation he/she wants.

4.1.1 Modified LAMMPS input file format

First and very importantly, comment out or delete the `run #` command in the LAMMPS input script. If this is not done, the LE driver will not work properly.

Secondly, there is a TEMPER tag. The syntax is somewhat strict, so follow as closely as possible and *do not* add extra spaces.

The TEMPER tag. This is the only tag used in the TEMPER code. Its syntax is:

```
#TEMPER: run [R], swap [S], temp [T], fix [F], seed [SS], dumpswap [D]
```

The variables in the square brackets are described here:

- [R] = An integer. The total number of MD steps to be taken.

- [S] = An integer. Swap frequency. This is the number of MD steps taken between exchange attempts.

- [T] = A float. Temperature for this replica. Makes sure it's the same as your NVT thermostat.

- [F] = A string. The fix ID of the NVT thermostat. Used to extract thermal information.

- [SS] = An integer. Seeds the random number generator for the Metropolis acceptance criteria. If [N] = 0, then the direction of swapping alternates as up/down. Otherwise, the direction of swaps is random. This is meant to be used for debugging purposes.

- [D] = An integer, either 1 or 0. This turns on/off if you want to have the file names for dumps exchanged between replicas. This is useful so that a given dump file contains only data from one temperature. Otherwise, it will be all mixed. It is recommended to set this to 1.

Sampling data can be extracted from TEMPER runs from dump files. It is best to set [D] to 1 so that the dump files contain data only for one temperature. The dump files will contain data according to the replica index.

4.2 REUS

Replica exchange umbrella sampling for MS-EVB with RAPTOR. Uses the fix_umbrella code for the umbrella bias.

4.2.1 Modified LAMMPS input file format

First and very importantly, comment out or delete the `run #` command in the LAMMPS input script. If this is not done, the LE driver will not work properly.

Secondly, there is a REUS tag. The syntax is somewhat strict, so follow as closely as possible and *do not* add extra spaces.

The REUS tag. This is the only tag used in the REUS code. Its syntax is (all on *one* line):

```
#REUS:   CVID [ID], run [R], swap [S], temp [T], fix [F], seed [SS],
coordtype [CT], short [SH], dump [D], dump_swap [DS], group_swap [GS]
```

Please ignore the line break after `[SS],`. I couldn't fit this all on one line, but the code expects only one line.

NOTE: There must be 3 spaces between `#REUS:` and `CVID`. This is very important! The driver code is strict about this.

The variables in the square brackets are described here:

- `[ID]` = A string. An identification you want to give to your replica to keep track of which it is. This is used to keep track of which COLVAR and log files we write to.

- `[R]` = An integer. The total number of MD steps to be taken.

- `[S]` = An integer. Swap frequency. This is the number of MD steps taken between exchange attempts.

- `[T]` = A float. Temperature for this replica. Makes sure it's the same as your NVT thermostat.

- `[F]` = A string. The fix ID for fix_umbrella. Used to extract bias information.

- `[SS]` = An integer. Seeds the random number generator for the Metropolis acceptance criteria. If `[N]` = 0, then the direction of swapping alternates as up/down. Otherwise, the direction of swaps is random. This is meant to be used for debugging purposes.

- `[CT]` = An integer. The coordinate type for fix_umbrella.
  The supported options are: 0 = COORD_CART and 2 = COORD_CYLINDER.

- `[SH]` = An integer. Number of short steps for asynchronous load balancer. This is an advanced option. Leave it as 0 if you don't know what this is. It is only useful if your MS-EVB replicas have disparate run times because they have different numbers of MS-EVB states. In that case, it is possible to set `[SH]` to a small integer, like 5, to take extra MD steps while waiting for slower replicas to catch up. This can boost the overall productivity by still running MD during MPI waits at exchange steps.

- `[D]` = An integer. Dump frequency. How often to write the bias information to the COLVAR.* files. `[D]` must satisfy the condition: $1 \leq$ `[D]` $\leq$ `[S]`. Otherwise, LE will crash.

- [DS] = An integer. 0 means turn off swapping dump file outputs. 1 or greater turns it on.

- [GS] = An integer. Controls whether or not the user wants REUS to swap the "Group 2" vector in fix_umbrella. In cases where the user has a curvilinear path of restraints, for example, the user would want to swap these restraints. On the other hand, if the user has a restraint based on center of mass of some atoms, the user should not swap the Group 2 restraints. 0 turns off group swapping. 1 or greater turns it on.

Bias information during the REUS run is output to COLVAR.* files, containing current bias energies and other information for each time step. These files only contain data from one umbrella window. The index of the communicator writing to the COLVAR file is listed as "comm" in the last column. V_bias is the bias energy at that time step.

At the end of an REUS run, final restart files are written to disk according to the final configuration of replica indices. That is, restart_final.2 corresponds to the umbrella window index 2. This is to make it easier to restart REUS runs without having to disentangle which replica index belongs to which restart file.

4.3   RELAMBDA

Replica exchange using off-diagonal scaling for MS-EVB. λ-Exchange.

4.3.1   Modified LAMMPS input file format

First and very importantly, comment out or delete the run # command in the LAMMPS input script. If this is not done, the LE driver will not work properly.

Secondly, there is a RELAMB tag (not RELAMBDA, no DA).

The RELAMB tag. This is the only tag used in the RELAMBDA code. Its syntax is (all on *one* line):

#RELAMB: LID [ID], run [R], swap [S], temp [T], fix [F], seed [SS], lambda [L]

The variables in the square brackets are described here:

- [ID] = A string. An identification you want to give to your replica to keep track of which it is. This is used to keep track of which COLVAR and log files we write to.

- [R] = An integer. The total number of MD steps to be taken.

- [S] = An integer. Swap frequency. This is the number of MD steps taken between exchange attempts.

- [T] = A float. Temperature for this replica. Makes sure it's the same as your NVT thermostat.

- [F] = A string. The fix ID for fix_umbrella. Used to extract bias information.

- [SS] = An integer. Seeds the random number generator for the Metropolis acceptance criteria. If [N] = 0, then the direction of swapping alternates as up/down. Otherwise, the direction of swaps is random. This is meant to be used for debugging purposes.

7

- [L] = A float. λ, the value by which to scale the off-diagonal coupling in MS-EVB. 1.0 is normal, and 0.0 turns off MS-EVB coupling, which is equivalent to vehicular hydronium transport. Setting λ much greater than 1.0 can be dangerous as it can over-couple waters and might make energy conservation worse. A recommended range is: $0 \leq \lambda \leq 1.5$, but you may find other values acceptable. Making $\lambda > 1$ should help to artificially enhance proton transport, although one should not collect statistics on any λ other than 1.0.

This replica exchange method is entirely untested for any real applications, and it is unpublished. Need to write paper? Run some calculations with this code and write about it!

4.4 COLVARX

Replica exchange umbrella sampling using the colvars library. Although multiple dimensions are documented below, COLVARX is only stable for one-dimensional REUS currently.

4.4.1 Modified LAMMPS input file format

First and very importantly, comment out or delete the `run #` command in the LAMMPS input script. If this is not done, the LE driver will not work properly.

There is a *mandatory* line in the input script for Hamiltonian exchange to work properly. The line is (put it at the bottom of your LAMMPS input script):

```
compute pe all pe
```

It simply adds a compute call for the potential energy to LAMMPS with the identification string "pe". LE looks for this compute in order to get the potential energy after swapping coordinates. Without it, LE will crash.

Thirdly, there are a number of LE tags to place at the top of the input file (not required to be at the top, but it is a good place for it). There are 4 relevant tag types, discussed in turn below. The syntax is somewhat strict, so follow as closely as possible and *do not* add extra spaces.

The COLVARX tag. This is the main tag that LE looks for in deciding what to do. The syntax is as follows:

```
#COLVARX: fix [F], seed [N]
```

The variables in the square brackets are described here:

- [F] = A string corresponding to the ID name of the colvar fix in this LAMMPS input script. If [F] = "none", then the driver does not search for the colvar fix. This is used for unconstrained Hamiltonian exchange or parallel tempering without colvars.

- [N] = An integer. Seeds the random number generator for the Metropolis acceptance criteria. If [N] = 0, then the direction of swapping alternates as up/down. Otherwise, the direction of swaps is random. This is meant to be used for debugging purposes.

The REPLICA tag. After the COLVARX tag, LE will search for the REPLICA tag to gather information specific to the given replica. Its syntax is:

```
#REPLICA: id [I], ndim [Ndim], temp [T], tdim [TD]
```

The variables in the square brackets are described here:

8

- [I] = An integer greater than or equal to zero. The lowest replica index must be zero and should increase by unit increments. That is, the range is 0 to (n-1), where n is the number of replicas. This is the index of this replica.

- [Ndim] = An integer corresponding to the number of dimensions in the multidimensional REUS run. Must be 1 or greater.

- [T] = A positive floating point number. The temperature of this replica. It should be the same as the temperature used later in the LAMMPS input script for the NVT ensemble, etc.

- [TD] = An integer. This is the index of the dimension on which to perform parallel tempering (i.e., temperature exchanges). This is optional, and it can only be for one dimension. If you don't want any parallel tempering, set this variable to -1, which informs LE to ignore it.

The DIMENSION tag. The next tag(s) to create describe how each exchange dimension should be set up. This is controlled via the DIMENSION tag. There are two options for this tag, depending on how you want to specify the number of swap attempts.
Option 1:

#DIMENSION: [D] num [NUM] run [RUN] swaps [SWAPS]

Option 2:

#DIMENSION: [D] num [NUM] run [RUN] swapfreq [SWAPFREQ]

You can have an arbitrary number of DIMENSION tags, each on their own line, as long as that number corresponds to [Ndim] for the REPLICA tag. The variables in the square brackets are described here:

- [D] = An integer greater than or equal to zero. The lowest replica index must be zero and should increase by unit increments. That is, the range is 0 to (n-1), where n is the number of replicas. This is the index of this dimension.

- [NUM] = An integer corresponding to the coordinate of this replica along this dimension. In range 0 to (x-1), where x is the number of replicas in this dimension.

- [RUN] = A positive integer. How many MD steps in total to take along this dimension.

- [SWAPS] = A positive integer. How many replica exchanges to attempt along this dimension throughout the simulation.

- [SWAPFREQ] = A positive integer. The frequency of swap attempts. How many MD steps are to be taken between swap attempts. This number must divide [RUN] (*i.e.* no remainder).

The DIMENSION tags must be consistent across all LAMMPS inputs for each replica. Otherwise, the run will fail.

The NEIGHBORS tag. Finally, you will need to make a NEIGHBORS tag for each DIMENSION tag, specifying which subcommunicators are to act as neighbors in the attempted replica exchanges. Neighbors are arbitrary, but if input incorrectly, LE will deadlock in communication. (Efforts are underway to automatically detect and report such errors.) The syntax for this tag is:

#NEIGHBORS: [D] [M] [P]

9

The variables in the square brackets are described here:

- [D] = An integer greater than or equal to zero. The lowest replica index must be zero and should increase by unit increments. That is, the range is 0 to (n-1), where n is the number of replicas. This is the index of the dimension being described.

- [M] = An integer. The minus direction neighbor subcommunicator index for this dimension. If it is negative, this means that this replica has no neighbor in the minus direction for this dimension. Otherwise, it must be 0 or greater in the range of subcommunicator indexes.

- [P] = An integer. The plus direction neighbor subcommunicator index for this dimension. If it is negative, this means that this replica has no neighbor in the plus direction for this dimension. Otherwise, it must be 0 or greater in the range of subcommunicator indexes.

It is possible to define circular dimensions by making appropriate NEIGHBOR tags. For example, for a set of three replicas (0, 1, 2, 3), we might have the following NEIGHBOR tags in each appropriate LAMMPS input script:

```
#NEIGHBORS: 0 3 1
#NEIGHBORS: 0 0 2
#NEIGHBORS: 0 1 3
#NEIGHBORS: 0 2 0
```

In order from replica 0 to replica 3, the above would create a circular dimension for four replicas.

Example of tags. Below is a possible example of the LE header for a LAMMPS input script for a one dimensional parallel tempering run with the same collective variable in each temperature:

```
#COLVARX: fix cv, seed 0
#REPLICA: id 2, ndim 1, temp 275.0, tdim 0
#DIMENSION: 0 num 2 run 500 swaps 5
#NEIGHBORS: 0 1 3
```

4.5   COORDX

Experimental coordinate exchange code. Not stable, but documented here anyway. Similar set up to COLVARX.

4.5.1   Modified LAMMPS input file format

First and very importantly, comment out or delete the run # command in the LAMMPS input script. If this is not done, the LE driver will not work properly.

Secondly, there is a *mandatory* line in the input script for Hamiltonian exchange to work properly. The line is (put it at the bottom of your LAMMPS input script):

```
compute pe all pe
```

It simply adds a compute call for the potential energy to LAMMPS with the identification string "pe". LE looks for this compute in order to get the potential energy after swapping coordinates. Without it, LE will crash.

Thirdly, there are a number of LE tags to place at the top of the input file (not required to be at the top, but it is a good place for it). There are 4 relevant tag types, discussed in turn below. The syntax is somewhat strict, so follow as closely as possible and *do not* add extra spaces.

The COORDX tag. This is the main tag that LE looks for in deciding what to do. COORDX is short for "Coordinate Exchange" because this code swaps the coordinates of each replica, making the code very generalizable for any sort of exchanges. The syntax is as follows:

```
#COORDX: fix [F], seed [N]
```

The variables in the square brackets are described here:

- [F] = A string corresponding to the ID name of the colvar fix in this LAMMPS input script. If [F] = "none", then the driver does not search for the colvar fix. This is used for unconstrained Hamiltonian exchange or parallel tempering without colvars.

- [N] = An integer. Seeds the random number generator for the Metropolis acceptance criteria. If [N] = 0, then the direction of swapping alternates as up/down. Otherwise, the direction of swaps is random. This is meant to be used for debugging purposes.

The REPLICA tag. After the COORDX tag, LE will search for the REPLICA tag to gather information specific to the given replica. Its syntax is:

```
#REPLICA: id [I], ndim [Ndim], temp [T], tdim [TD]
```

The variables in the square brackets are described here:

- [I] = An integer greater than or equal to zero. The lowest replica index must be zero and should increase by unit increments. That is, the range is 0 to (n-1), where n is the number of replicas. This is the index of this replica.

- [Ndim] = An integer corresponding to the number of dimensions in the multidimensional REUS run. Must be 1 or greater.

- [T] = A positive floating point number. The temperature of this replica. It should be the same as the temperature used later in the LAMMPS input script for the NVT ensemble, etc.

- [TD] = An integer. This is the index of the dimension on which to perform parallel tempering (i.e., temperature exchanges). This is optional, and it can only be for one dimension. If you don't want any parallel tempering, set this variable to -1, which informs LE to ignore it.

The DIMENSION tag. The next tag(s) to create describe how each exchange dimension should be set up. This is controlled via the DIMENSION tag. There are two options for this tag, depending on how you want to specify the number of swap attempts.
Option 1:

```
#DIMENSION: [D] num [NUM] run [RUN] swaps [SWAPS]
```

Option 2:

```
#DIMENSION: [D] num [NUM] run [RUN] swapfreq [SWAPFREQ]
```

11

You can have an arbitrary number of DIMENSION tags, each on their own line, as long as that number corresponds to [Ndim] for the REPLICA tag. The variables in the square brackets are described here:

- [D] = An integer greater than or equal to zero. The lowest replica index must be zero and should increase by unit increments. That is, the range is 0 to $(n$-1$)$, where n is the number of replicas. This is the index of this dimension.

- [NUM] = An integer corresponding to the coordinate of this replica along this dimension. In range 0 to $(x$-1$)$, where x is the number of replicas in this dimension.

- [RUN] = A positive integer. How many MD steps in total to take along this dimension.

- [SWAPS] = A positive integer. How many replica exchanges to attempt along this dimension throughout the simulation.

- [SWAPFREQ] = A positive integer. The frequency of swap attempts. How many MD steps are to be taken between swap attempts. This number must divide [RUN] (*i.e.* no remainder).

The DIMENSION tags must be consistent across all LAMMPS inputs for each replica. Otherwise, the run will fail.

The NEIGHBORS tag. Finally, you will need to make a NEIGHBORS tag for each DIMENSION tag, specifying which subcommunicators are to act as neighbors in the attempted replica exchanges. Neighbors are arbitrary, but if input incorrectly, LE will deadlock in communication. (Efforts are underway to automatically detect and report such errors.) The syntax for this tag is:

#NEIGHBORS: [D] [M] [P]

The variables in the square brackets are described here:

- [D] = An integer greater than or equal to zero. The lowest replica index must be zero and should increase by unit increments. That is, the range is 0 to $(n$-1$)$, where n is the number of replicas. This is the index of the dimension being described.

- [M] = An integer. The minus direction neighbor subcommunicator index for this dimension. If it is negative, this means that this replica has no neighbor in the minus direction for this dimension. Otherwise, it must be 0 or greater in the range of subcommunicator indexes.

- [P] = An integer. The plus direction neighbor subcommunicator index for this dimension. If it is negative, this means that this replica has no neighbor in the plus direction for this dimension. Otherwise, it must be 0 or greater in the range of subcommunicator indexes.

It is possible to define circular dimensions by making appropriate NEIGHBOR tags. For example, for a set of three replicas (0, 1, 2, 3), we might have the following NEIGHBOR tags in each appropriate LAMMPS input script:

#NEIGHBORS: 0 3 1
#NEIGHBORS: 0 0 2
#NEIGHBORS: 0 1 3
#NEIGHBORS: 0 2 0

In order from replica 0 to replica 3, the above would create a circular dimension for four replicas.

Example of tags. Below is a possible example of the LE header for a LAMMPS input script for a one dimensional parallel tempering run with the same collective variable in each temperature:

```
#COORDX: fix cv, seed 0
#REPLICA: id 2, ndim 1, temp 275.0, tdim 0
#DIMENSION: 0 num 2 run 500 swaps 5
#NEIGHBORS: 0 1 3
```

4.6 MREUS

Multi-dimensional replica exchange umbrella sampling. Any combination of REUS, TEMPER, and/or RELAMBDA. Can also be used in place of the above REUS, TEMPER, and/or RELAMBDA routines, if you like. This is the coolest replica exchange routine, and it potentially can scale over many, many processors.

4.6.1 Modified LAMMPS input file format

First and very importantly, comment out or delete the `run #` command in the LAMMPS input script. If this is not done, the LE driver will not work properly.

Secondly, there are a number of LE tags to place at the top of the input file (not required to be at the top, but it is a good place for it). There are 4 relevant tag types, discussed in turn below. The syntax is somewhat strict, so follow as closely as possible and *do not* add extra spaces.

The MREUS tag. This is the main tag that LE looks for in deciding what to do. The syntax is as follows:

```
#MREUS: evb [EVB], fix [FIX], thermo [THERMO], seed [SS],
coordtype [CT], short [SH], dump [D], dump_swap [DS], group_swap [GS]
```

Please ignore the line break after `[SS],`. I couldn't fit this all on one line, but the code expects only one line.

The variables in the square brackets are described here:

- `[EVB]` = A string corresponding to the ID name of the EVB fix in this LAMMPS input script. If `[EVB]` = "none", then the driver does not search for the EVB fix.

- `[FIX]` = A string corresponding to the ID name of the fix_umbrella fix in this LAMMPS input script. If `[FIX]` = "none", then the driver does not search for the fix_umbrella fix.

- `[THERMO]` = A string corresponding to the ID name of the NVT fix in this LAMMPS input script. If `[THERMO]` = "none", then the driver does not search for the NVT fix.

- `[SS]` = An integer. Seeds the random number generator for the Metropolis acceptance criteria. If `[SS]` = 0, then the direction of swapping alternates as up/down. Otherwise, the direction of swaps is random. This is meant to be used for debugging purposes.

- `[CT]` = An integer. The coordinate type for fix_umbrella.
  The supported options are: 0 = COORD_CART and 2 = COORD_CYLINDER.

- [SH] = An integer. Number of short steps for asynchronous load balancer. This is an advanced option. Leave it as 0 if you don't know what this is. It is only useful if your MS-EVB replicas have disparate run times because they have different numbers of MS-EVB states. In that case, it is possible to set [SH] to a small integer, like 5, to take extra MD steps while waiting for slower replicas to catch up. This can boost the overall productivity by still running MD during MPI waits at exchange steps.

- [D] = An integer. Dump frequency. How often to write the bias information to the COLVAR.* files. [D] must satisfy the condition: $1 \leq [D] \leq [S]$. Otherwise, LE will crash.

- [DS] = An integer. 0 means turn off swapping dump file outputs. 1 or greater turns it on.

- [GS] = An integer. Controls whether or not the user wants REUS to swap the "Group 2" vector in fix_umbrella. In cases where the user has a curvilinear path of restraints, for example, the user would want to swap these restraints. On the other hand, if the user has a restraint based on center of mass of some atoms, the user should not swap the Group 2 restraints. 0 turns off group swapping. 1 or greater turns it on.

The REPLICA tag. After the COLVARX tag, LE will search for the REPLICA tag to gather information specific to the given replica. It has two acceptable syntaxes, which include or exclude the λ information:

#REPLICA: id [I], ndim [Ndim], temp [T](, lambda [LAMBDA])

The part in parentheses is optional. Do not include parentheses in your input. The variables in the square brackets are described here:

- [I] = An integer greater than or equal to zero. The lowest replica index must be zero and should increase by unit increments. That is, the range is 0 to $(n\text{-}1)$, where n is the number of replicas. This is the index of this replica.

- [Ndim] = An integer corresponding to the number of dimensions in the multidimensional REUS run. Must be 1 or greater.

- [T] = A positive floating point number. The temperature of this replica. It should be the same as the temperature used later in the LAMMPS input script for the NVT ensemble, etc.

- [TD] = An integer. This is the index of the dimension on which to perform parallel tempering (i.e., temperature exchanges). This is optional, and it can only be for one dimension. If you don't want any parallel tempering, set this variable to -1, which informs LE to ignore it.

- [LAMBDA] = A float. λ, the value by which to scale the off-diagonal coupling in MS-EVB. 1.0 is normal, and 0.0 turns off MS-EVB coupling, which is equivalent to vehicular hydronium transport. Setting λ much greater than 1.0 can be dangerous as it can over-couple waters and might make energy conservation worse. A recommended range is: $0 \leq \lambda \leq 1.5$, but you may find other values acceptable. Making $\lambda > 1$ should help to artificially enhance proton transport, although one should not collect statistics on any λ other than 1.0.

The DIMENSION tag. The next tag(s) to create describe how each exchange dimension should be set up. This is controlled via the DIMENSION tag. There are two options for this tag, depending on how you want to specify the number of swap attempts.
Option 1:

```
#DIMENSION: [D] num [NUM] type [TYPE] run [RUN] swaps [SWAPS]
```

Option 2:

```
#DIMENSION: [D] num [NUM] type [TYPE] run [RUN] swapfreq [SWAPFREQ]
```

You can have up to 3 of DIMENSION tags, each on their own line, as long as that number corresponds to [Ndim] for the REPLICA tag. The variables in the square brackets are described here:

- `[D]` = An integer greater than or equal to zero. The lowest replica index must be zero and should increase by unit increments. That is, the range is 0 to $(n-1)$, where n is the number of replicas. This is the index of this dimension.

- `[NUM]` = An integer corresponding to the coordinate of this replica along this dimension. In range 0 to $(x-1)$, where x is the number of replicas in this dimension.

- `[TYPE]` = Integer from set: 1 = REUS, 2 = TEMPER, 3 = RELAMBDA. Describes what type of dimension this is. Cannot have more than 1 of any type of dimension currently.

- `[RUN]` = A positive integer. How many MD steps in total to take along this dimension.

- `[SWAPS]` = A positive integer. How many replica exchanges to attempt along this dimension throughout the simulation.

- `[SWAPFREQ]` = A positive integer. The frequency of swap attempts. How many MD steps are to be taken between swap attempts. This number must divide `[RUN]` (*i.e.* no remainder).

The DIMENSION tags must be consistent across all LAMMPS inputs for each replica. Otherwise, the run will fail.

The NEIGHBORS tag. Finally, you will need to make a NEIGHBORS tag for each DIMENSION tag, specifying which subcommunicators are to act as neighbors in the attempted replica exchanges. Neighbors are arbitrary, but if input incorrectly, LE will deadlock in communication. (Efforts are underway to automatically detect and report such errors.) The syntax for this tag is:

```
#NEIGHBORS: [D] [M] [P]
```

The variables in the square brackets are described here:

- `[D]` = An integer greater than or equal to zero. The lowest replica index must be zero and should increase by unit increments. That is, the range is 0 to $(n-1)$, where n is the number of replicas. This is the index of the dimension being described.

- `[M]` = An integer. The minus direction neighbor subcommunicator index for this dimension. If it is negative, this means that this replica has no neighbor in the minus direction for this dimension. Otherwise, it must be 0 or greater in the range of subcommunicator indexes.

- `[P]` = An integer. The plus direction neighbor subcommunicator index for this dimension. If it is negative, this means that this replica has no neighbor in the plus direction for this dimension. Otherwise, it must be 0 or greater in the range of subcommunicator indexes.

It is possible to define circular dimensions by making appropriate NEIGHBOR tags. For example, for a set of three replicas (0, 1, 2, 3), we might have the following NEIGHBOR tags in each appropriate LAMMPS input script:

```
#NEIGHBORS: 0 3 1
#NEIGHBORS: 0 0 2
#NEIGHBORS: 0 1 3
#NEIGHBORS: 0 2 0
```

In order from replica 0 to replica 3, the above would create a circular dimension for four replicas.
Example of tags. Below is a possible example of the LE header for a LAMMPS input script for parallel tempering in dimension 1 and REUS in dimension 0:

```
#MREUS: evb evb, fix fes, thermo 1, seed 0, coordtype 0, short 0, dump 5, dump_swap 0, group_swap 0
#REPLICA: id 2, ndim 2, temp 310.0, lambda 1.00
#DIMENSION: 0 num 2 type 1 run 100 swaps 5
#NEIGHBORS: 0 -1 3
#DIMENSION: 1 num 2 type 2 run 100 swaps 10
#NEIGHBORS: 1 0 -1
```

4.7   Calling the LE driver executable

The call to the binary executable `ens_driver` has a few command line options. The general call looks like this:

```
ens_driver P [-suffix omp] [-log] -readinput (infile)
```

The parts in square brackets are optional, but the parts in parentheses are mandatory variables. The variable descriptions are here:

- `ens_driver` = The binary executable filename.

- `P` = A positive integer. The number of replicas (*i.e.*, the number of input files).

- `[-suffix omp]` = Optional flag for running LAMMPS with OpenMP multithreading. Recommended to turn on.

- `[-log]` = Flag to turn on writing output to log files for each replica. Otherwise, no log files are written. Turning this will produce lots of text and may become a disk space issue if not careful.

- `(infile)` = The file containing the list of LAMMPS input script files and how many processors to use for each replica.

The `infile` is a text file and has the following syntax:

```
[Subcommunicator index] [LAMMPS script filename] [# MPI ranks for subcomm] (Optional # partitions)
```

There should not be anything else in this text file. The number of MPI ranks for each subcommunicator must sum to the number of MPI ranks in MPI_COMM_WORLD. You may need to specify the full path for each LAMMPS input script.

The optional number of partitions is primarily intended for use with the state decomposition algorithm for MS-EVB, which can make MS-EVB simulations with RAPTOR much faster if there are several MS-EVB states. Usually, 2 or 4 partitions works best, but you will need to test this on your own. Note that each partition will have an equal number of MPI ranks, as determined by dividing [Number of MPI ranks for this subcomm] by (Optional number of partitions). The code will crash with an error if there is an inconsistent number of MPI ranks and/or partitions. If the number of partitions is left out, it is assumed that no partitioning is being used.

The above executable, of course, will need to be launched as part of an MPI run. This means that the user will need to use a command like

```
mpiexec -np 8 ens_driver 2 -suffix omp -readinput foo.bar
```

to launch 8 MPI processes with the LE driver split into 2 replicas, with input files described in file foo.bar. Finally, note that you will want to set the Unix/Linux shell environment variable OMP_NUM_THREADS to the appropriate number of CPU cores prior to your run in order to take advantage of the multithreading parallelism.

4.8 Output

By default, no LAMMPS log files nor LAMMPS screen files are created during a REUS run to cut down on disk space usage. However, writing to the LAMMPS log file can be turned on (see Section 4.7), which can be useful in debugging.

The colvars library will write output to various files, and you should make sure that each is labelled correspondingly to the LAMMPS inputs for each subcommunicator. COLVARS files are written by the REUS and MREUS routines.

4.9 Examples

There are a handful of examples provided in the directory examples/. Try them out!