5-2018

Implementation of Replica Exchange with Dynamic Scaling in GROMACS 2018

Gregory John Schwing
University of New Orleans

Implementation of Replica Exchange with Dynamic Scaling in GROMACS 2018

An Honors Thesis

Presented to

The Department of Computer Science

Of the University of New Orleans

In Partial Fulfillment

of the Requirements for the Degree of

Bachelor of Science, with Departmental Honors

in Computer Science

by

Gregory John Schwing

May 2018

Acknowledgement

I would like to first thank my family for their love and support: Adrian, Diane, Mary Claire, Michael, Patrick, Nonie, and Prec. I would like to thank my friend, Nathan Walker, for helping me debug the toughest of bugs, teaching me C++, and serving as the alpha tester for the README's. Manish Bhatt began this work in completion of his Master's in Computer Science, and the groundwork he laid was instrumental in my prompt completion. For this I thank him. Lepaul Love tirelessly performed data analysis of the optimization process, and I am indebted to him for this. I would like to thank Dr. Rick for making many walks over to the Summa lab, and when we found a bug before he even arrived, his patience as we sent him back across campus. Finally, I would like to thank Dr. Christopher Mark Summa for his selfless dedication to my success. He is a great man.

Foreword

This thesis can get quite technically overwhelming at some points, so I invite the reader to retreat two pages back and begin rereading when feeling lost. I have taken great care to maintain locality of ideas and a continuous development of logic. This thesis represents my shedding of the comfortable presentation that biology presents concerning proteins, as I pull back the veil and face the statistical physics head on. "No one said it would be easy. No one said it would be this hard." –The Scientist, Coldplay

List of Figures

Abstract

This is a problem of sampling. The number of classical states of an N-body system grows with O(3 ^ N). To sample this space, advanced techniques are required. Replica Exchange (RE), also known as parallel tempering, is an example that uses parallelization, and Hamiltonian Replica Exchange is a subset of RE that scales the energy of the replicas. The number of simulations required grows at O(N^(1/2)), where N is number of atoms in the system. Replica Exchange with Dynamical Scaling (REDS) attempts to address this problem to decrease computational cost. It has been shown to increase efficiency 10-fold. We implemented REDS in GROMACS 2018. (Abraham 2015)

All changes to the source code were written in the form of parallel methods. Scripts were written in Python and Perl to automate the experiment entirely. An exchange connects a region of high energy space, far above the surface of the landscape, to low energy space, which approaches the surface of the landscape, which represents the natural conformational progression of the molecule. Using REDS we were able to achieve exchanges at temperatures spaced too far apart to exchange using normal RE. Ergo, the flexibility of dynamical scaling allowed regions of phase space that would have gone unsampled to be mapped, addressing our initial problem of sampling.

Keywords: Molecular Dynamics, Computational Biophysics, Monte Carlo, Replica Exchange, Parallel Programming

Table of Contents

Introduction

Molecular dynamics (MD) is a computer simulation method for studying the trajectories of atoms and molecules. The field was first introduced in the 1950's by theoretical physicists using the MANIAC I, one of the earliest computers (Fermi 1955; Alder and Wrainwright 1959; Rahman 1964). Then twenty years later, Levitt and Warshel proved MD could be applied to biological macromolecules by publishing the first simulation of a protein, known as bovine pancreatic trypsin inhibitor, in 1975 (Levitt and Warshel 1975). Biologists turned to MD seeking a physical explanation for the cooperativity of the hemoglobin molecule. The theoretical model proposed by Monod, Wyman, and Changeaux, coupled with the solving of the structure of hemoglobin by Perutz presented the first opportunity for MD to demonstrate its role in the scientific community (Monod, Wyman, and Changeux 1978; Perutz et al. 1960; Perutz 1960).

Building upon the work of Levitt et al, Case and Karplus simulated an Oxygen molecule binding to myoglobin, a structural relative of hemoglobin, and the results were quantitatively agreeable with experimental kinetic values (Case and Karplus 1979). This increased desire among biologists to utilize MD, however, a prerequisite is knowledge of protein structure, a formidable undertaking, in some cases impossible to acquire. The most effective method to obtain a protein structure is x-ray crystallography. The conditions in which proteins crystallize are drastically different from natural conditions,

which results in discrepancy between the native structure and simulation conformation. Major strides were made by Michael Levitt to improve MD by simulating in explicit water, as opposed to in a vacuum (Levitt and Sharon 1988). From this work, the field of protein refinement arose, seeking to further define the forces, via energy functions, responsible for the deformation from the native state. Important advancements, such as the work done by Summa and Levitt using pairwise atomic potentials, as opposed to traditional energy functions, have led to more accurate protein refinement (Summa and Levitt 2007). However, two formidable problems still remain with MD-based structure refinement, the first being assurance that the native state is the global minimum, and the second being verification of adequate conformational sampling to reach the native state (Feig 2017). In this project, we focused on the latter through implementation of a technique known as Replica Exchange with Dynamic Scaling (REDS) in the GROMACS software package.

In order to understand dynamic scaling and Replica Exchange, also known as parallel tempering, one must have a decent understanding of the tenets of statistical mechanics (see Appendix 1 – Background), specifically the Ergodic Hypothesis of Thermodynamics, taken from the work of Ludwig Boltzmann (Boltzmann 1896). According to Patrascioiu, the Ergodic Hypothesis states, "the time average value of an observable—which of course is determined by the dynamics—is equivalent to an ensemble average, that is, an average at one time over a large number of systems all of which have identical thermodynamic properties but are not identical on the molecular

level" (Patrascioiu 1987). However, the Kolmogorov–Arnold–Moser (KAM) theorem places a constraint on the Ergodic Hypothesis stating a dynamic system may enter quasiperiodic motion, as opposed to ergodic, if insufficient perturbation is provided (Kolmogorov 1954; Moser 1962; Arnol 1963).

One of the limitations of molecular dynamics is the failure of the theory of a priori probabilities translating into practice. One of the fundamental assumptions when using molecular dynamics is that initial conditions and length of simulation should be independent of results. The reason this fails in practice is that certain microstates will never occur for various reasons. This is known as a lack of ergodicity in a simulation. This problem usually arises because a simulation is stuck in a local minimum of phase space. One method to overcome the lack of ergodicity is to introduce a parallel simulation of the same replica at a greater temperature which provides the requisite energy to escape the minima. This technique is known as parallel tempering, or Replica Exchange. Earl et al provides a succinct definition of the technique with an illustrative figure:

> The general idea of parallel tempering is to simulate M replicas of the original system of interest, each replica typically in the canonical ensemble, and usually each replica at a different temperature. The high temperature systems are generally able to sample large volumes of phase space, whereas low temperature systems, whilst having precise sampling in a local region of phase space, may become trapped in

local energy minima during the timescale of a typical computer simulation. Parallel tempering achieves good sampling by allowing the systems at different temperatures to exchange complete configurations. Thus, the inclusion of higher temperature systems ensures that the lower temperature systems can access a representative set of low-temperature regions of phase space (Earl and Deem 2005).

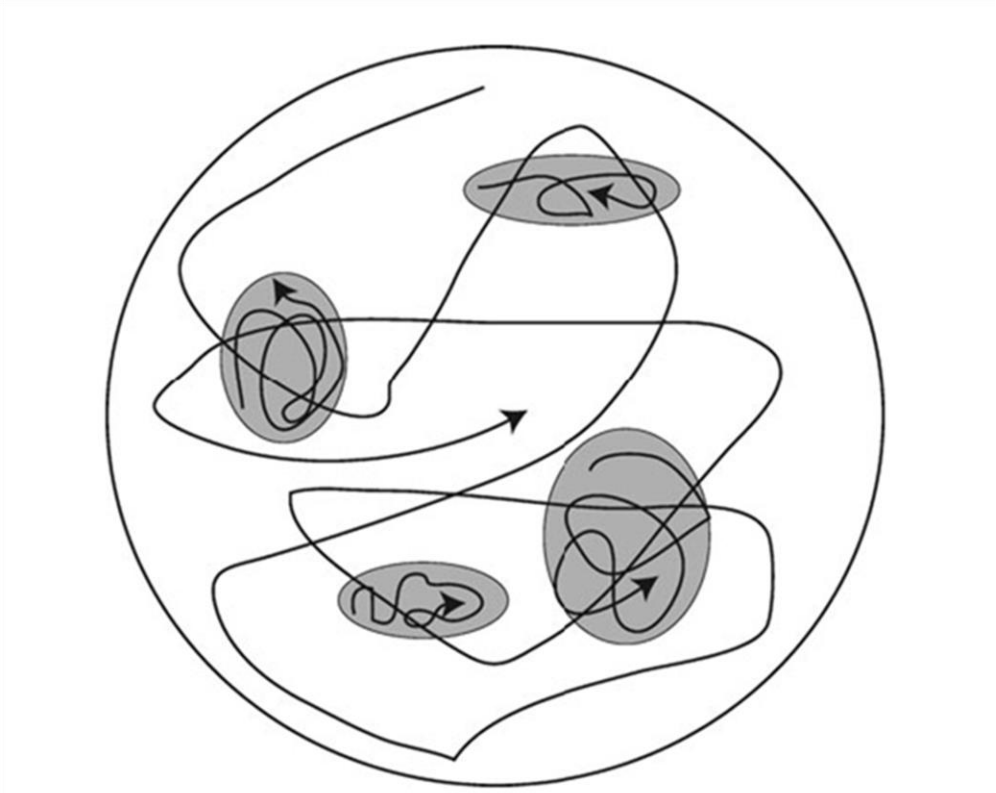


Figure 1. Replica Exchange Walking in Phase Space, (Earl and Deem 2005)

This figure illustrates the lower temperature replicas (grey ovals) becoming trapped in local minima, and the higher temperature replicas (roaming arrow) exploring phase space.

Replica exchange attempts to ensure equiprobability of microstates through the use of several replicas of the system, simultaneously and independently simulated at different temperatures. Explained in by Ostermeier et al, "Pairs of replicas usually close in temperature are exchanged with a specific Metropolis transition probability …. allow[ing] conformations that are trapped in locally stable states at a low simulation temperature to escape by exchanging with replicas at higher simulation temperature" (Ostermeir and Zacharias 2013). Replica exchange attempts to address two limitations in MD, one computational and one physical. The first is the short time scale over which computers are able to model peptides. Replica exchange allows one to extend the time scale and increase the accuracy of the ensemble average. The second limitation is the constraint placed by the KAM theorem, which replica exchange handles through simulation at a high enough temperature to ensure ergodicity (Earl and Deem 2005). The success of replica exchange depends on the nature (enthalpic vs entropic), magnitude (energy barrier height vs thermal energy per degree of freedom), and landscape (double well vs "golf course") of the energy barriers, with documented cases of both improving and worsening of the MD model (Ostermeir and Zacharias 2013; Zuckerman and Lyman 2006; Machta 2009; Nymeyer 2008; Denschlag, Lingenheil, and Tavan 2008).

The ensemble of a replica exchange simulation consists of many temperatures, but the simulations do not interact energetically. The partition function is shown below:

Equation 1.1 – Partition Function of Replica Exchange (Earl and Deem 2005).

$$Q = \prod_{i=1}^{M} \frac{q_i}{N!} \int d\boldsymbol{r}_i^N \exp[-\beta_i U(\boldsymbol{r}_i^N)],$$

$$\text{where } q_i = \prod_{j=1}^{N} (2\pi m_j k_B T_i)^{3/2}$$

The formula for calculating exchange probabilities between replica I and J are shown below:

Equation 1.2 – Delta for Replica Exchange (Earl and Deem 2005).

$$\Delta_{nm} = -(\beta_i - \beta_j)\left(U(r_i{}^N) - U(r_j{}^N)\right)$$

The definition of this delta is subject to change as we progress to more advanced sampling techniques. Regardless of how the delta is calculated, exchange acceptance is always evaluated according to a Metropolis Criterion:

Equation 1.3 – The Metropolis Criterion (Earl and Deem 2005).

$$P_{n\leftrightarrow m} = \begin{cases} 1 & if\ \Delta_{nm} \le 0 \\ e^{-\Delta nm} & if\ \Delta_{nm} > 0 \end{cases}$$

Swaps are attempted between replica i and j. The condition for acceptance of a standard replica exchange swap is a Monte-Carlo Metropolis Criterion which evaluates the product of the differences between the two thermodynamics betas and the potential

energies, U(r), as shown above in Equation 1.12. When a swap occurs the positions, r, of the atoms are exchanged between the parallel simulations at different temperatures, and the replicas continue exploring phase space from their new positions. This process is known as replica exchange.

Unfortunately, the order of growth of required number of replicas is O(N^1/2), where N is number of atoms. As the molecule grows in size, it necessitates a number of simulations to evaluate all possible conformations. The energy distributions of each replica must overlap to successfully exchange, illustrated below.

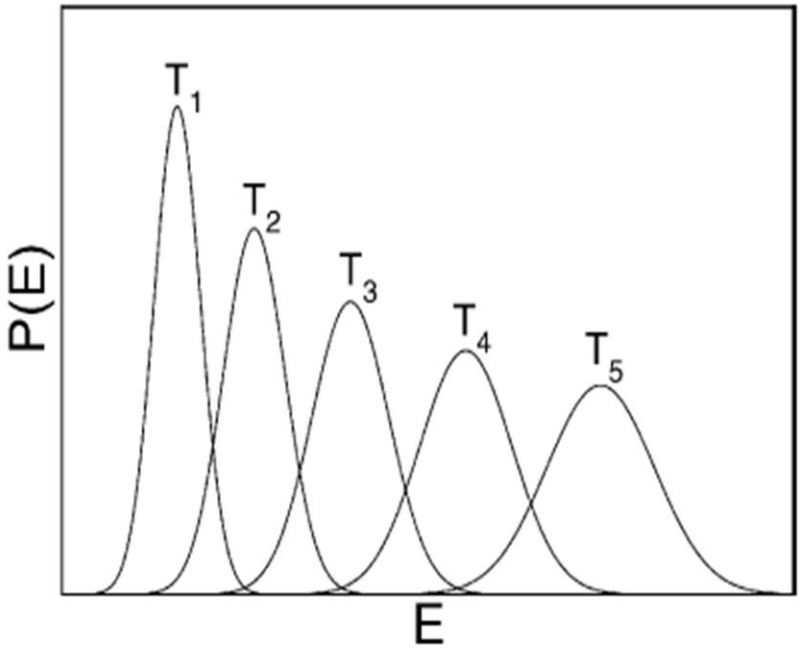


Figure 2. Standard Replica Exchange Energy Distributions, (Earl and Deem 2005).

This figure illustrates the shapes of the energy distributions of a standard replica exchange simulation. The X-axis is Energy; the Y-axis is Probability of a system being found at that energy. Each bell curve represents a replica at a different temperature.

When considering a biological molecule, which is on the order of thousands of atoms, this quickly becomes prohibitive. To overcome this restriction, scientists have developed modifications to the replica exchange method. One of the most effective, and the type this project falls under, is Hamiltonian Replica Exchange. Hamiltonian Replica Exchange scales part of all of the potential, or in other words flattens and widens the curves. Exchanges are then attempted between normal unscaled replicas and the scaled replicas. This action is based on the premise that we are reducing the barriers between exchange, particularly the energy difference between replicas. There are many ways to scale the potential, for example you may scale only certain forces that are used to generate the potential, such as the Van Der Waals interactions, or transform the entire potential using Tsallis Scaling (Torrie and Valleau 1977; Berg and Neuhaus 1992). The equation for a standard Hamiltonian Replica Exchange Delta is given below:

Equation 1.4 – Delta for Hamiltonian Replica Exchange (Rick 2007).

$$\Delta_{nm} = -\beta_m[U_m(r_n) - U_m(r_m)] + \beta_n[U_n(r_n) - U_n(r_m)]$$

Dynamical scaling is an optimization of Hamiltonian replica exchange, which works by scaling the replicas with a dynamic variable, known as zeta, ζ, varying from 0 to 1. An exchange will effectively be forced as zeta approaches an endpoint, either 0 or 1, because the Boltzmann weighting of the scaled replica approaches the weighting of one of the neighboring unscaled replicas (Bhatt and Rick 2015). Replica scaling can in some cases increase efficiency ten-fold (Rick 2007). Each replica possesses its own value of zeta. A

single-variable, third-degree biasing equation is also added to each replica to coerce zeta into distributing normally across the domain from 0 to 1. The coefficients of this equation are iteratively tuned to find appropriate values. We appropriately call this technique Replica Exchange with Dynamical Scaling (REDS). The process is similar to the replica exchange technique described above, but Exchanges between replicas I and J are accepted according to a biased metropolis criterion and are scaled dynamically, shown in blue.

Equation 1.5 – Potential Energy Functions for REDS (Rick 2007).

$$U_m(\zeta, r) = \frac{[\boldsymbol{\zeta} * \boldsymbol{\beta_{mR}} + (\mathbf{1} - \boldsymbol{\zeta})\boldsymbol{\beta_{mL}}]}{\boldsymbol{\beta_m}} * U(r) + {E_{bias}}^m(\zeta)$$

$$U_n(\zeta, r) = \frac{[\boldsymbol{\zeta} * \boldsymbol{\beta_{nR}} + (\mathbf{1} - \boldsymbol{\zeta})\boldsymbol{\beta_{nL}}]}{\boldsymbol{\beta_n}} * U(r) \ + {E_{bias}}^n(\zeta)$$

Equation 1.6 – Delta for REDS (Rick 2007).

$$\Delta_{nm} = (\beta_{nR} - \beta_{mR})\big(\zeta_n * U(r_n) - \zeta_m * U(r_m)\big) - (\beta_{nL} - \beta_{mL})\big((1 - \zeta_n) * U(r_n) - (1 - \zeta_m) * U(r_m)\big) - \beta_n * ({E_{bias}}^n(\zeta_n) - {E_{bias}}^n(\zeta_m)) - \beta_m * ({E_{bias}}^m(\zeta_m) - {E_{bias}}^m(\zeta_n))$$

The additional computation is justified by the resulting energy distributions which are illustrated below.

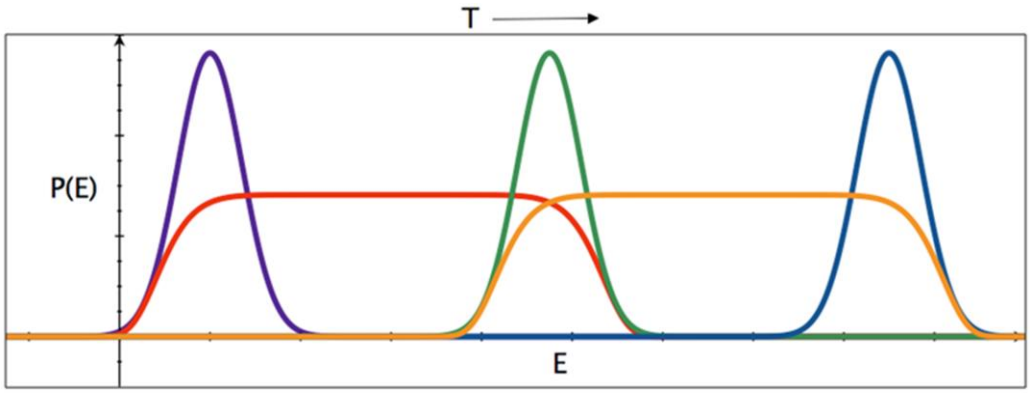


Figure 3. REDS Energy Distributions, (Rick 2007).

This figure illustrates the shapes of the energy distributions of a REDS simulation. The X-axis is Energy; the Y-axis is Probability of a system being found at that energy. Each bell curve represents a replica at a different temperature. The flattened curves are the shapes of scaled replicas. Their oblong shape allow for greater gaps between standard replicas.

Zeta is changed according to a Monte-Carlo Metropolis Criterion along the potential energy functions given in equation 1.5. The algorithm is detailed below.

For replica m:

Equation 1.7 – Repeated Equation 1.5 (Rick 2007).

$$U_m(\zeta, r) = \frac{[\zeta * \beta_{mR} + (1-\zeta)\beta_{mL}]}{\beta_m} * U(r) + {E_{bias}}^m(\zeta)$$

Attempt a move:

Equation 1.8 – Calculating the new zeta (Rick 2007).

$$\zeta_{new} = \zeta + \delta\zeta(random - 0.5)$$

"*random*" is a random number between 0 and 1 and $\delta\zeta$ is move size. This is a standard Monte Carlo move. Accept the move based on the energy difference:

Equation 1.9 – Delta for Standard Monte-Carlo (Rick 2007).

$$\Delta U_m = U_m(\zeta_{new}, r) - U_m(\zeta, r)$$

$$= \frac{[\zeta_{new} * \beta_{mR} + (1 - \zeta_{new}) * \beta_{mL}]}{\beta_m} * U(r) + {E_{bias}}^m(\zeta_{new}) - (\frac{[\zeta * \beta_{mR} + (1 - \zeta) * \beta_{mL}]}{\beta_m} * U(r) + {E_{bias}}^m(\zeta))$$

$$= (\zeta_{new} - \zeta)\frac{(\beta_{mR} - \beta_{mL})}{\beta_m} * U(r) + {E_{bias}}^m(\zeta_{new}) - {E_{bias}}^m(\zeta)$$

This delta is evaluated against a Metropolis Criterion shown below, Equation 1.3:

$$P_{n\leftrightarrow m} = \begin{cases} 1 & if\ \Delta_{nm} \le 0 \\ e^{-\Delta nm} & if\ \Delta_{nm} > 0 \end{cases}$$

Materials and Methods

This work is built upon the most up-to-date version of GROMACS, commit hash # : 2566608adc423653a2b60fc559da4223057ca253 . Various features were introduced, with a majority of the changes in the *repl_ex.cpp* file . This software was built upon the existing replica exchange feature of the "mdrun" program. Parallel methods were written to mirror the original stack trace. The two versions are only partially parallel, converging where possible and diverging where necessary. The flow chart of the program illustrates where they diverge.

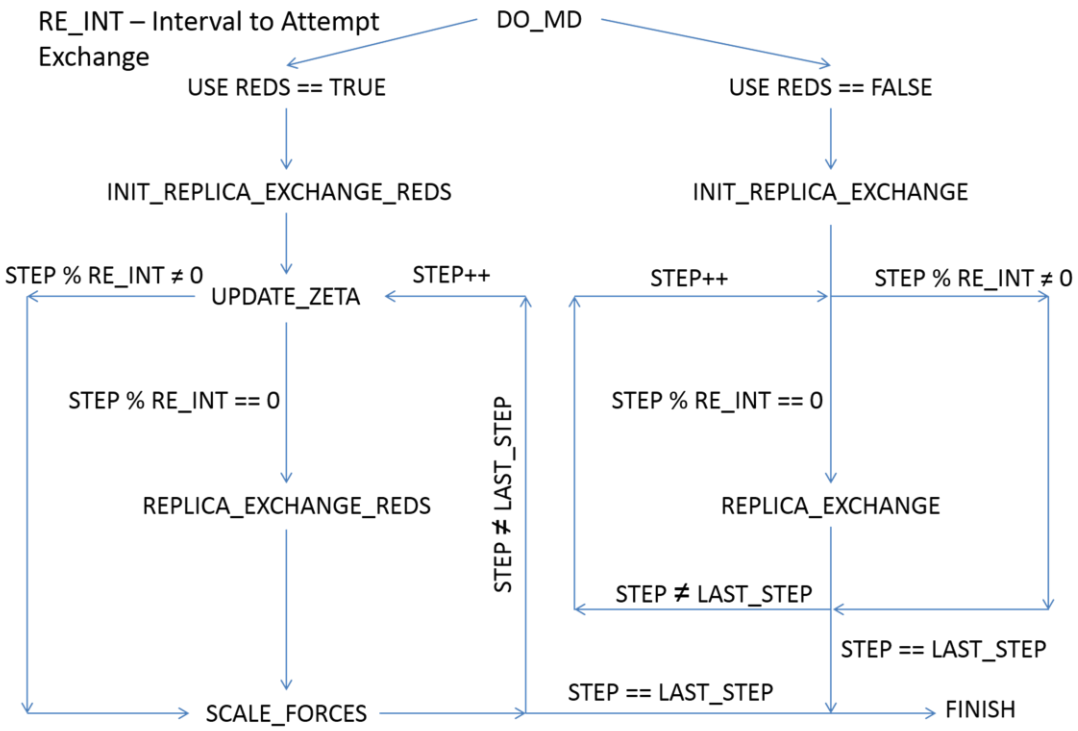


Figure 4, Flow Chart of Program

This figure illustrates the flow of control of a single process in a multi-process simulation. It also demonstrates the differences between the code we implemented (Left) and the original code base (Right).

The addition of the -reds command line flag was accomplished by adding a checking for the string, “reds = true”, in the “mdp” file, which stands for Molecular Dynamics Parameters. This value is initialized to false by default and overwritten in the case it is found. The beginning of every REDS-specific method chain is enclosed in a conditional checking the value of this boolean. This design rescued the original program from alteration during the development process. The parallel methods were written in the same files as their counterparts, to avoid issues with building and installing. Preparing a REDS simulation is identical to its counterpart, with the added option to specify biasing coefficients in the parameter file. These biasing coefficients allow the user to tune the distribution of the dynamical variable, zeta, to uniformity.

The replica exchange methods are called from the molecular dynamics main method, DO_MD (Appendix A2.1), at an interval specified by the user on the command line. Before entering this loop, N replica objects are initialized, where N corresponds to the number of simultaneous simulations, by the INIT_REPLICA_EXCHANGE_REDS (Appendix A2.2) method. The REDS initialization process differs from normal replica exchange only in the additional allocation of two dynamic arrays to hold the biasing coefficients, a 2D array, and zetas, a 1D array, of each replica. After these objects are initialized, the method returns control to the DO_MD method. This is the main method of each process simulation, and it normally consists of a for loop that integrates the equations of motion and updates the potions of the atoms. We added method calls to

UPDATE_ZETA (Appendix A2.3) and SCALE_FORCES (Appendix A2.4), both of which are called each iteration of the loop.

Each replica possesses a state object which holds the < x, y, z > coordinates of the atoms and the value of zeta, along with many other properties. This state object is what is swapped when an exchange is accepted; therefore, we always retrieve the current value of zeta from the state object prior to calling UPDATE_ZETA (Appendix A2.3) to ensure we aren't using an outdated value of zeta, in the occasion a swap has occurred. This freshly retrieved zeta is passed into UPDATE_ZETA (Appendix A2.3) which alters it by a small, random amount and then tests it for acceptance by Monte-Carlo Metropolis Criterion. Since UPDATE_ZETA(Appendix A2.3) is called every step, it was a convenient place to do output, so we split the method in two parts – the Monte-Carlo walker and the output to file, which is primarily used for optimization purposes.

When UPDATE_ZETA is called, the zeta returned from a call to getZeta(state), is passed as an argument. The replica object's copy of zeta is immediately overwritten with this value. Then the Monte-Carlo walker changes zeta, and if the Metropolis Criterion accepts the change, the replica objects copy will be once again overwritten with this new zeta value. When UPDATE_ZETA returns, it returns the replica object's value of zeta. The next line of code is a call to setZeta(returnedZeta, state). This was designed this way because replicas exchange state objects, so it was easier to add zeta to the state object and allow it to tag along during the exchange process than rewrite the complex

MPI exchange code. However, we need the global array of zetas, necessary when testing for exchanges, is generated from the replica object's zetas. Our design maintains consistency between the two copies of zeta, one belonging to replica object the other to the state object, even in the case of an exchange.

When UPDATE_ZETA (Appendix A2.3) returns, control is returned to the DO_MD (Appendix A2.1) method which proceeds to integrate the equations of motion, calculate the potential energy of the molecule, and calculate the force on each atom. However, before applying the forces to the atoms and allowing them to move, we scale the forces by calling SCALE_FORCES (Appendix A2.4). This is necessary as we intend to scale the energy distributions to promote exchanges. Since force is the derivative of energy and one may isolate constants from an expression when taking a derivative, we must multiply the force by the same constant we plan to multiply the energy. When SCALE_FORCES (Appendix A2.4) returns, DO_MD (Appendix A2.1) proceeds to apply the scaled forces to the atoms and update their positions accordingly.

Finally, if it is an exchange attempt interval, the REPLICA_EXCHANGE_REDS method will be called. This method proceeds to call three methods – PREPARE_TO_DO_EXCHANGE, TEST_FOR_REPLICA_EXCHANGE_REDS (Appendix A2.5) , and EXCHANGE_STATE. During the pre-production optimization of the biasing coefficients, REPLICA_EXCHANGE_REDS is actually called every step, but terminated prematurely, immediately prior to the EXCHANGE_STATE method call.

This was setup because a side effect of TEST_FOR_REPLICA_EXCHANGE_REDS (Appendix A2.5) is the generation and propagation of the global array of zetas to all processes. During optimization we need this global array every step for parameterization of the bias coefficients. This approach allowed the avoidance of writing a second method to generate these global arrays, with only a small time sacrifice. This sacrifice is only present in the short optimization runs before a longer production run. During the production simulation, when time is much more precious, REPLICA_EXCHANGE is only called on exchange intervals.

The TEST_FOR_REPLICA_EXCHANGE_REDS (Appendix A2.5) method has two main purposes – to generate and propagate the global arrays of bias coefficients and zetas, and as its name suggests, to perform the calculations to determine whether or not an exchange should occur. It is necessary for each replica object to know the value of all replicas' biasing coefficients and zeta values, suggesting some type of shared memory space between replicas. The biasing coefficients are static throughout the simulation, while the zeta variables are updated on every step of MD-loop. A message passing method is called on the bias coefficients only on the first call to this method, generating the global array of the static biasing coefficients, which is then read-only for the rest of the run. This illustrates a key distinction between bias coefficients and zeta. Bias coefficients are an attribute of the environment in which the replica is simulated, along with temperature. The zetas are attributes of the replica, along with the atoms themselves, and is swapped between different environments. To account for this

difference, the message passing method is called to share the dynamic zeta values every time TEST_FOR_REPLICA_EXCHANGE_REDS (Appendix A2.5) is called. This occurs at the beginning of the method.

When determining whether or not an exchange should occur, only the even or odd replicas are eligible for exchange on a given step. Which one, even or odd, alternates each method call. The schematic following the entry (Appendix A2.5) illustrates which replicas are eligible each time TEST_FOR_REPLICA_EXCHANGE_REDS is called. With the most up to date version of all the zeta variables and a fully populated biasing coefficient 2D array, the CALC_DELTA_REDS (Appendix A2.6) method is called on the even, or odd, replicas which calculates a four part sum. The CALC_DELTA_REDS (Appendix A2.6) method uses the zetas, betas , and biasing coefficients of each of the two replicas passed as parameters to calculate this sum. The scaling logic is integrated into this calculation by simply setting the beta variables to all be equal if it meets any of the following criteria – first, last, or an odd-numbered replica. The same calculation is performed on both scaled and unscaled replicas, but if the replica is unscaled, two of the betas cancel out. This four-part sum is returned as the delta that is evaluated by metropolis criterion to determine the acceptance or rejection of the exchange. The ineligible replicas, determined by being the opposite parity of the MD step divided by exchange interval, modulus divided by 2, are automatically rejected. The results of these evaluations are written to an array of boolean values indicating whether or not an acceptance occurred in that replica. Interestingly enough, each simultaneous process

determines the acceptance or rejection of each the replica being simulated, whilst only possessing the ability to initiate exchanges from whichever one it represents.

Once the TEST_FOR_REPLICA_EXCHANGE_REDS (Appendix A2.5) method returns with its array of booleans, bExchanged, it calls PREPARE_TO_DO_EXCHANGE. This reinitializes helper arrays for performing the exchanges and calculates the cyclic decomposition of the swaps. Once PREPARE_TO_DO_EXCHANGE returns, the master thread will perform the exchanges by calling EXCHANGE_STATE, which is a thread-safe atomic swap of the state objects of two processes. Recall that the state holds the positions of the atoms and the zeta value of the replica. This concludes the method calls of a replica exchange. This process is repeated until the simulation ends.

Once the simulation ends, the value of zeta, the first derivative of the potential energy with respect to zeta, the theoretical value of first derivative of potential energy with respect to zeta when zeta is 0, and theoretical value of the first derivative of potential energy with respect to zeta when zeta is 1 are printed to a file. This file is used in the optimization of the values of the biasing coefficients : a, b, and c. The optimization process works as follows:

Since the bias doesn't directly act on the coordinates, we temporarily ignore the bias. Without the bias, the potential energy is:

Equation 2.1 – Unbiased, dynamically scaled Potential Energy, (Rick 2007).

$$U_m(\zeta, r) = \frac{[\zeta * \beta_{m1} + (1-\zeta)\beta_{m2}]}{\beta_m} * U(r)$$

When zeta is 1 the potential is:

Equation 2.1 – Unbiased, dynamically scaled Potential Energy, zeta = 1, (Rick 2007).

$$U_m(1, r) = \frac{[1 * \beta_R + (1-1)\beta_L]}{\beta_m} * U(r)$$

$$= \frac{\beta_R}{\beta_m} * U(r)$$

When you calculate a Boltzmann factor, you multiply it by β_m, "my Beta," and exponentiation it. Therefore the Boltzmann factor when zeta is 1 is:

Equation 2.3 – Boltzmann Factor, when zeta = 1, (Rick 2007).

$$U_m(1, r) = \beta_m * \frac{\beta_R}{\beta_m} * U(r)$$

$$= \cancel{\beta_m} * \frac{\beta_R}{\cancel{\beta_m}} * U(r)$$

$$= \beta_R * U(r)$$

Notice the β_m's cancel out. This is the same Boltzmann factor of an unscaled replica at the right temperature. We use the Boltzmann factor of the right replica as our theoretical value for our replica when zeta at 1, giving us a right boundary for our parameterization. We then take the derivative to obtain the theoretical value of the first derivative of

potential energy with respect to zeta. The same process, except on the left temperature, is repeated to obtain the theoretical value when zeta is 0.

For the scaled replicas we want the average force on zeta to be zero for all zeta.

Equation 2.4 – Average Force on Zeta, set equal to zero, (Rick 2007).

$$-\frac{dE}{d\zeta} = 0$$

The average force on zeta is defined as – (dE / dz):

Equation 2.5 – Average Force on zeta, in terms of zeta, beta, and U, (Rick 2007).

$$-\frac{dE}{d\zeta} = -\frac{\beta_L - \beta_R}{\beta_m} * U(r) - a - 2b * \zeta - 3c * \zeta^2$$

We move the response variable to opposite side of the equation of the bias equation:

Equation 2.6 – Final form of Polynomial Regression fitting zeta to average force, (Rick 2007).

$$-\frac{\beta_L - \beta_R}{\beta_m} * U(r) = a + 2b * \zeta + 3c * \zeta^2$$

We perform polynomial regression of zeta onto the response variable. We use the theoretical values of zeta at 0 and 1 as boundary conditions. This returns values for the coefficients a, b, and c. These coefficients are then used as the biasing coefficients in the next pre-production run. This is repeated until the distribution of zeta is uniformly distributed between 0 and 1. Then a production run is initiated.

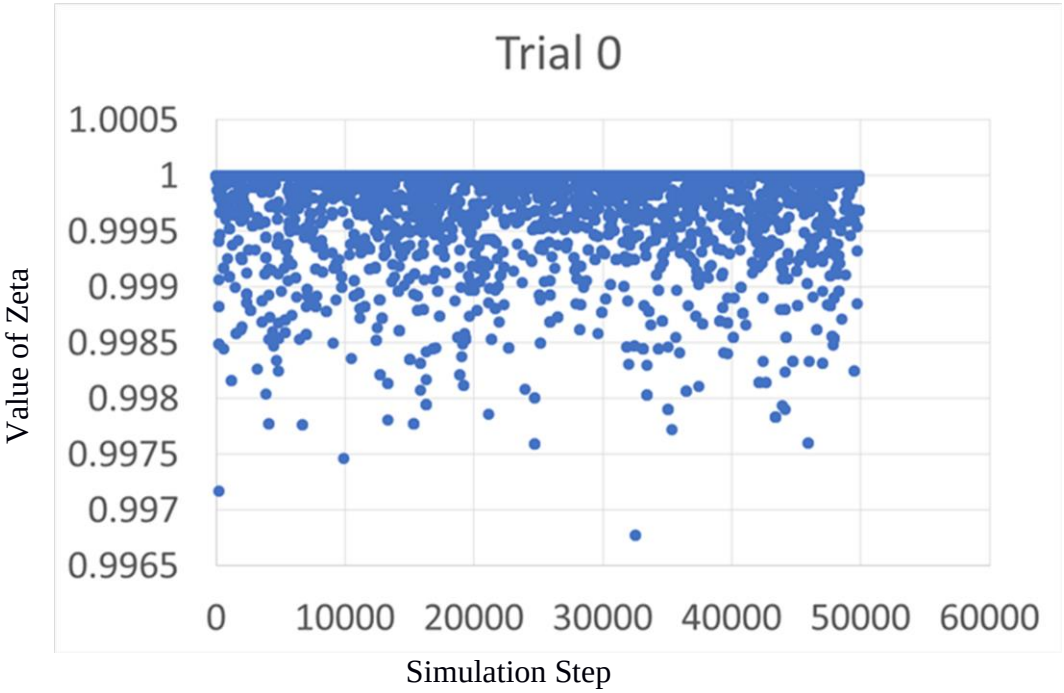
Results

The MCMC algorithm potentially alters the zeta value every step of the simulation. The value of each replica's zeta is printed to a file each step, which is used to optimize the bias coefficients by fitting equations 2.5 and 2.6. The process of fitting and rerunning the simulation was performed four times, and the effect on the distribution of zeta is noticeable. In trial 0, the values of zeta span a range of about 0.001, from 1.00 to 0.996 . In trials 1 and 2, the values of zeta sample across the entire domain, from 0 to 1, but sample zero far more than any other number. In trial 3, the zeta values do not span the entire domain, limited from 1 to 0.6; however, they are much more evenly distributed across this subset of the domain than before.

In a perfect trial, the zeta values would exhibit even sampling across the entire domain from 0 to 1. These values of zeta influence how often the program exchanges, with exchanges taking place when zeta is close to 0 and 1. We were able to obtain a pseudonormal distribution between 0 and 1, after eight trials of optimization.

Figure 5, Zeta Optimization

X-Axis : Step
Y-Axis : Value of Zeta



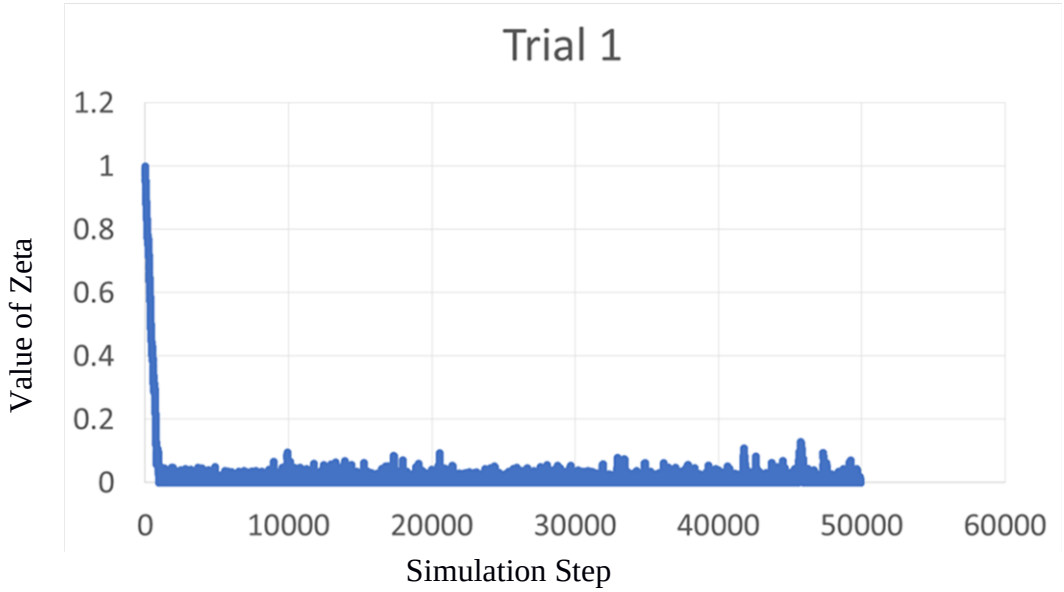
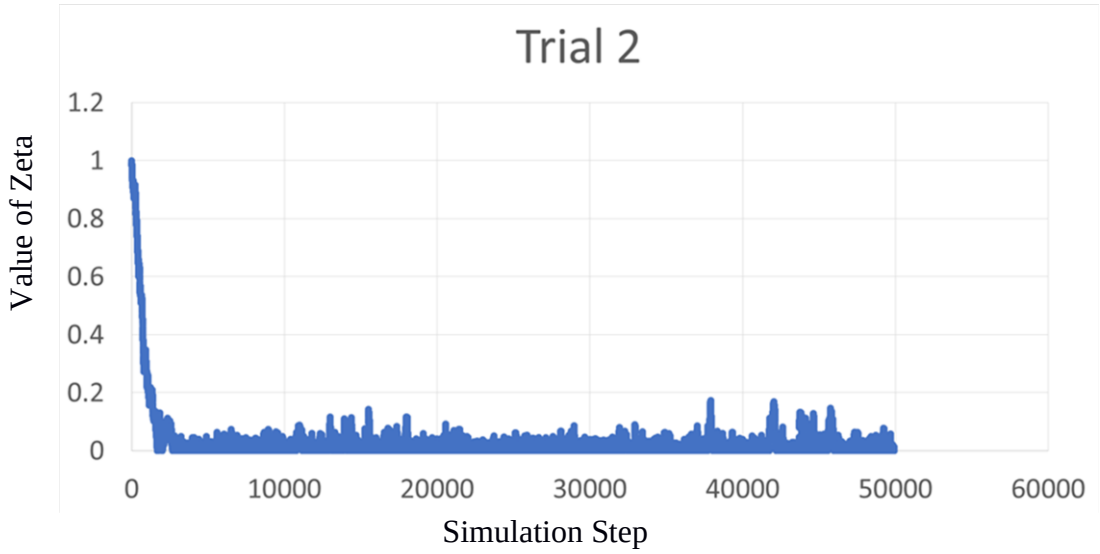


Figure 5 cont.,

X-Axis : Step
Y-Axis : Value of Zeta



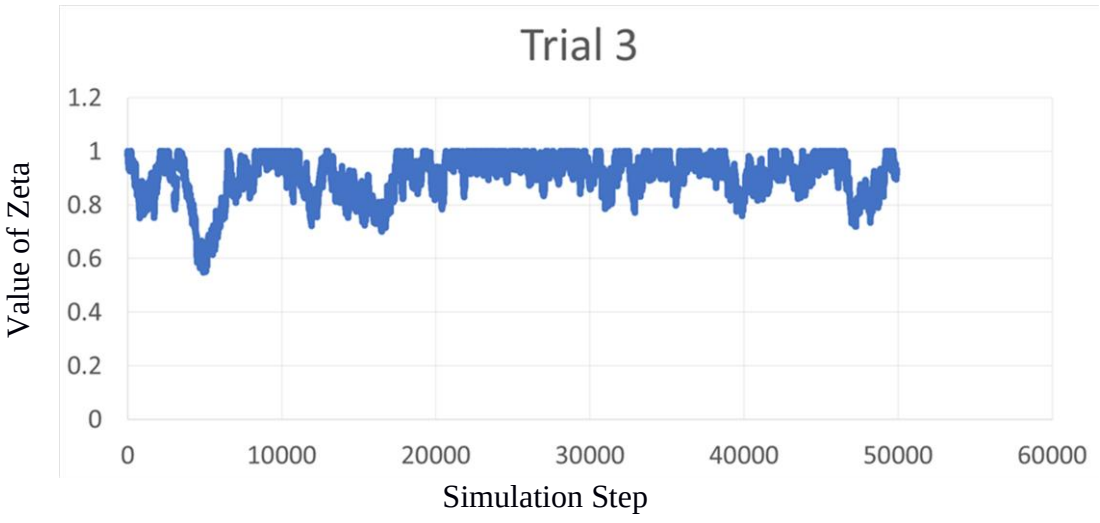
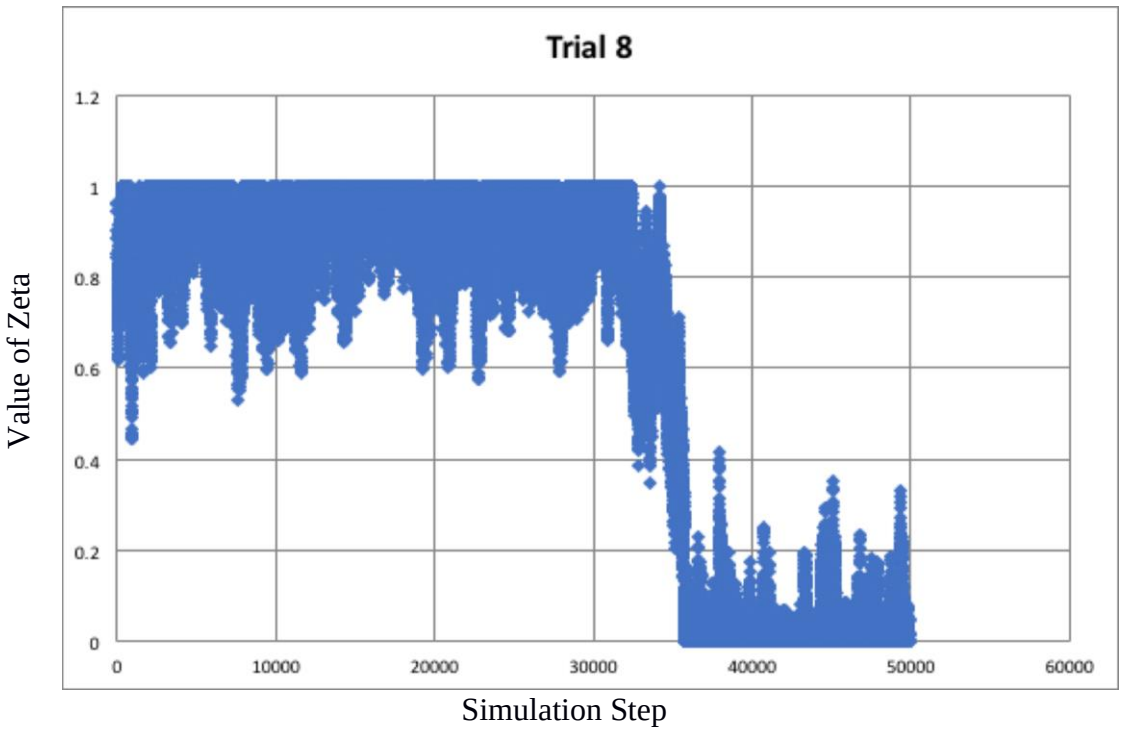


Figure 5 cont.,

X-Axis : Step
Y-Axis : Value of Zeta



Discussion

A very thorough README is included to guide the user through the process of downloading and installing the code (Appendix 3). We have written two scripts to automate the process of building, one for the user with GPU acceleration and for without. Once the user has built GROMACS-REDS, then the program may be run by following the README in Appendix 4. There are scripts for running both replica exchange and REDS. This allows for the user to easily compare both of the methods. The data presented in the previous section was generated on the simulation of a dialanine peptide. One may compare the energy landscapes generated by standard replica exchange and REDS to illustrate the power of this approach. The effectiveness of this tool has been shown previously, and for more in depth data analysis of a REDS simulation, refer to Rick et al (Rick 2007; Lee and Rick 2009).

After building GROMACS-REDS, the program is used identically to RE. However, when the mdrun program is invoked, REDS checks for a new command line argument, -reds, and if present, attempts MCMC on zeta every step, scales forces with that zeta every step, and calculates potential energy difference with a dynamically scaled equation. Upon a successful exchange, zetas are swapped, along with conformations, between the neighboring replicas. As of now, only nearest-neighbor RE is supported, but this is a limitation of production GROMACS not of our patch. There is still the issue of zeta not truly sampling evenly from 0 to 1. It is a pseudonormal distribution, and we would like it to traverse from zero to one more often than once per fifty thousand steps. This problem

may be solved by more optimization trials, but this is a known problem with using a bias. There are many other variants of Monte-Carlo that could be tried to obtain more uniform values of zeta, but for now we label this is as a limitation of our study.

Until this project, Dr. Rick, of the UNO Department of Chemistry, had only implemented REDS in the MD program AMBER, written in Fortran, which has less support for parallelism, and one of his students, Manish Bhatt, had implemented it in GROMACS 4 (Bhatt and Rick 2015). The objective of this project was threefold 1) Implement REDS in GROMACS 2018, 2) Develop it alongside the original Replica Exchange Software to allow for easy comparison, and 3) Create a user-friendly scripting environment to encourage widespread adaptation. This project was novel as Dr. Rick is the inventor of REDS, and our implementation in GROMACS 2018 is the first to ever be done. Our future plans include testing REDS1 on intricate biological molecules, optimizing the program for increased efficiency, and working to get REDS merged upstream with the production GROMACS branch.

Appendix 1 – Background

Consider a liter of salt water. The liter can be described by physical properties such as temperature, pressure, and volume, obtained by measuring the liter with a thermometer, manometer, and graduated cylinder. These properties are, therefore, called measurables or observables. When a scientist uses a thermometer to measure the temperature, he will most likely take multiple measurements and then average them to find a value of greater confidence. This is necessary for two reasons – systematic error and random error. The systematic error arises due to the limited accuracy of a thermometer, which we assume is not accurate enough to detect changes at a very small scale, or in other words a microscopic scale. The second source of error, random error, arises due to the nature of temperature as a thermodynamic property. According to the McGraw-Hill Dictionary of Scientific & Technical Terms, a thermodynamic property is, "A quantity which is either an attribute of an entire system or is a function of position which is continuous and does not vary rapidly over microscopic distances, except possibly for abrupt changes at boundaries between phases of the system; examples are temperature, pressure, volume, concentration, surface tension, and viscosity. Also known as macroscopic property" (McGraw-Hill 2003). The random error will exist as long as temperature is measured by thermometer, since macroscopic properties are functions of the system, and not limited to whatever position the thermometer happens to be placed. To obtain a temperature measurement void of random error, one must evaluate the entire system at once, obtaining an absolute value.

While this project resides firmly in the realm of statistical physics, the basic expressions of statistical physics are more easily explained with terms coined by quantum physicists. According to Professor McClure of Portland State, "Statistical Mechanics is a statistical approach to solving the classical n body problem in order to study the same bulk properties of matter as thermodynamics but doing so at the microscopic level" (McClure 2010). Now let's reconsider the same liter of water, but at the microscopic level. It consists of Na+ and Cl- ions solvated by H2O molecules. Each one of these molecules possesses a position and momentum, which can be used to solve differential equations that describe the macroscale behavior of the system. The general form of this equation is below, where G is an observable thermodynamic property.

Equation A1.1 – A time average, (McClure 2010).

$$\overline{G} = G_{obs} = \frac{1}{\tau}\int_{t_0}^{\tau+t_0} G[p(t), q(t)]\, dt$$

For this is a time average to be valid, tau need be long enough to be independent of G. This equation will be solved for each molecule in the system.

Each one of these molecules can exist at a number of different quantum mechanical states, represented by Ψ, which a function of position and time. The collection of all the Ψ forms the microstate of the system, Ω, the product of all the Ψ. Each Ω possesses a quantum mechanical average similar to thermodynamic properties such as temperature,

pressure, and volume. The expression for the value of this quantum mechanical average, devoid of any random error, is shown below.

Equation A1.2 – Quantum Mechanical Average, (McClure 2010).

$$G_{obs} = \overline{G} = \int \Omega^* \hat{G} \Omega \, d\tau$$

Every measurable in a physical system is associated with a quantum mechanical operator, and the Hamiltonian, H, is the operator associated with the system energy, E, a scalar, real value since the Hamiltonian is a Hermitian operator (Nave 2005).

Physical Quantity		*Operator in Coordinate Representation*	*Operator in Momentum Representation*
Total energy	E	$-\frac{\hbar^2}{2m}\nabla^2 + V(\mathbf{r})$	$\frac{p^2}{2m} + V(i\hbar\nabla_{\mathbf{p}})$

Figure 6. Quantum Hamiltonian Operator

This figure illustrates the operator that determines the system energy of its operand. It returns the sum of the kinetic and potential energy.

The Hamiltonian is the operator shown above, the Ω is the product of all the Ψ in the system, and E is a scalar, real value. By solving the below equation, one identifies the Ω corresponding to an energy. In molecular dynamics, we know the Ω and wish to calculate the E.

Equation A1.3 – The Dirac Formalism of the Hamiltonian

$$H_s|\ \Omega> = E|\ \Omega>,$$

(Bra-Ket notation of the Solving of the N-body Hamiltonian)

The individual terms can be calculated as shown below.

Equation A1.4 – The State of a Quantum System, (McClure 2010).

$$\Omega = \prod_{i=1}^{n} \Psi_i(q,t)$$

Equation A1.5 – The Hamiltonian of the System, (McClure 2010).

$$\widehat{H}_s = \prod_{i=1}^{n} \mathrm{H}_i(p,q,t)$$

Unfortunately there are many solutions to the above equation, where many Ω correspond to the same value of E. This phenomenon, a many-to-one mapping, is known as degeneracy. The degeneracy of an N-body system grows at O(10^N), where N is number of particles (McQuarrie 1973). This means, for all but the smallest systems at the lowest energies, the number of Ω a system can be in at any given time is astronomical, and the system is equally likely to be in any of its Ω(E) eigenstates (Frenkel and Smit 2002). It is, however, possible to determine the value of a quantum mechanical property of a system, but to do so we appeal to statistical physics.

To do so, we must first identify all the Ω that are consistent with a few parameters, such as energy, number of particles, and volume. From this subset, we will calculate the value of the mechanical property using equation 2. Then, the unweighted average of the property across the many Ω is calculated. Finally, as McQuarrie states, "We then *postulate* that this average mechanical property corresponds to a parallel thermodynamic property. For example, we postulate that the average energy corresponds to the thermodynamic energy" (McQuarrie 1973). We will now introduce some concepts, such as ensemble and phase space, to formalize the properties of this unweighted average across Ω, most notably time-independence.

First introduced by Gibbs in 1902, an ensemble is a collection of size A composed of Ω's with identical macroscopic properties, such as number of particles, N, volume, V, and total energy, E. This implies that the ensemble consists of A*N particles, A*V volume, and A*E total energy (McQuarrie 1973). One key subtlety to note is that A*E is defined as total energy. If total energy is held constant, as opposed to the energy of each individual microstate, then a problem is introduced. There are infinitely many ways to sum up a collection of energy values given an arbitrary total energy. Fortunately, we can restrict the domain of energy values to the energy eigenvectors of the Schrodinger equation, which are finite and correspond to the given values of N and V, limiting our possible values of E to these eigenvectors, E_j.

$$\begin{pmatrix} A_{11} & A_{12} & \cdots & A_{1n} \\ A_{21} & A_{22} & \cdots & A_{2n} \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ \cdot & \cdot & \cdots & \cdot \\ A_{n1} & A_{n2} & \cdots & A_{nn} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{pmatrix} = \lambda \begin{pmatrix} x_1 \\ x_2 \\ \cdot \\ \cdot \\ \cdot \\ x_n \end{pmatrix} \qquad (10.12)$$

Figure 7. Eigenvector, (McQuarrie 1973).

This figure illustrates the properties of an eigenvector. Notice that the two **x**'s are equal. A is an n x n matrix. **x** is the eigenvector. λ is a scalar number. The product of the two matrices on the left results in a scalar multiple of **x** on the right.

An eigenvector according to the Oxford dictionary is "[a] vector which when operated on by a given operator gives a scalar multiple of itself" (Oxford Dictionary) Therefore, our domain of Ω is defined as those microstates found in $\Omega(E \in E_j)$. Note that the use of Ω as an operator, like so $\Omega(E_j)$, returns the number of microstates consistent with the given parameter, in this case a certain value of Energy. As noted above, there is a degeneracy here with $\Omega(E_j)$ being greater than one. We impose the principle of equal a priori probabilities, where each of these degenerate microstates is equally likely, so that the density in phase space is directly correlated to the degree of degeneracy,

The spectrum of allowed energy levels are given by the set of E_i that solve following equation.

Equation A1.6 – Repeat of Equation A1.3 with E_i constrained to eigenstates, (Frenkel and Smit 2002).

$$H \mid \Omega > = E_i \mid \Omega >,$$

The mathematical foundation of degeneracy of an isolated N-body system, according to Merzbacher, “is represented mathematically by the Hamiltonian for the system having more than one linearly independent eigenstate with the same eigenvalue” (Merzbacher 1970).

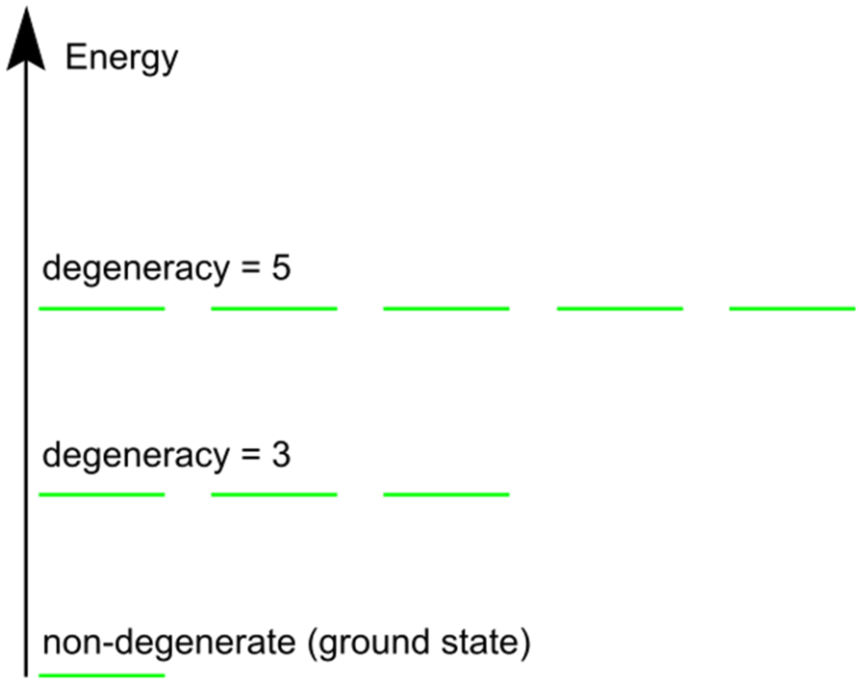


Figure 8. A visualization of Degeneracy, (Wikipedia Contributors 2018).

Each blue line represents an eigenstate, **i**, at an Energy level, E_i. Each of the eigenstates on a given level are equally likely to occur in nature from a statistical point of view.

All these axioms and definitions allow for an interesting property emerge from ensembles, "In an ensemble of mechanical systems identical in nature and subject to forces determined by identical laws, but distributed in phase in any continuous manner, the density-in-space is constant in time for the varying phases of a moving system; provided, that the forces of a system are functions of its coordinates, either alone or with the time" (Gibbs 1902). This theorem separates the associated properties of an ensemble from time, accepting the average value across all possible Ω – a quantity known as an ensemble average. The phase and density-in-space that Gibbs referred to are a

probability density function (PDF) with an x-axis of some thermodynamic property and y-axis representing the probability of finding the system at that value of x and the value of f(x), respectively. The integral of this PDF represents the ensemble average and is known as the partition function and can be thought of as the sum over all the states. This is the unweighted average we proposed earlier. Note that the arbitrary thermodynamic property is represented by G.

Equation A1.8 – Integral Representation of the Partition Function, (Frenkel and Smit 2002).

$$\langle G \rangle = \frac{\int G_{\mu}(\boldsymbol{r}_1,\ldots\boldsymbol{r}_{N_m}) e^{-\beta U(\boldsymbol{r}_1,\ldots\boldsymbol{r}_{N_m})} d\boldsymbol{r}_1,\ldots\boldsymbol{r}_{N_m}}{\int e^{-\beta U(\boldsymbol{r}_1,\ldots\boldsymbol{r}_{N_m})} d\boldsymbol{r}_1,\ldots\boldsymbol{r}_{N_m}}$$

We now introduce the first bridge from theory to computation by presenting the computable equivalent of the partition function.

Equation A1.9 – Integral Representation of the Partition Function, (Frenkel and Smit 2002).

$$\langle G \rangle = \frac{1}{M} \sum_{\mu=0}^{M} G_{\mu}(\boldsymbol{r}_1, \ldots \boldsymbol{r}_{N_m})$$

M represents the number of measurements one takes, which is essentially the number of simulation steps one runs on a computer, and G is some measurable. This calculation became readily possible with the introduction of electric computing machines in the 1950's and sure enough was one of the first uses for the ENIAC (Fermi 1955). It is a requirement that the value of G be independent of the number of measurements in a simulation, the M value. These are often not independent in practice, and the answer to this problem is a technique known as replica exchange. Replica exchange is nothing more than running multiple simulations of the same system at different temperatures simultaneously, but to fully comprehend the power this affords us we must present a formal derivation of thermodynamic beta.

The intrinsic thermodynamic properties, such as temperature, volume, and pressure, can be defined by using the appropriate operator; Figure 6 lists only one of many, and the partition function. However, extrinsic properties, such as entropy, require further derivation. Consider a system with total energy E, with a pair of weakly interacting subsystems, able to exchange energy.

The total energy is represented by,

Equation A1.10 – Weakly interacting systems (Frenkel and Smit 2002).

$$Etot = E1 + E2$$

There are many different ways to distribute energy across the two systems, and each distribution corresponds to a total number of degenerate states of the system.

Equation A1.11 – Number of degenerate states given a choice of E1, X represents multiplication (Frenkel and Smit 2002).

$$\#\ of\ degenerate\ states = \Omega 1(E1)\ X\ \Omega 2(E2)$$

Equation A1.12 – An additive formula of the degeneracy, (Frenkel and Smit 2002).

$$ln\ \Omega\ (E1,\ Etot\text{-}E1) = ln\ \Omega 1(E1) + ln\ \Omega 2(Etot\text{-}E1)$$

A logical step is to solve for the value of E1 which maximizes the value of Eq 1.4.

Equation A1.13 – Condition for the Maximum of 1.5 (Frenkel and Smit 2002).

$$\frac{\partial ln\Omega(E1, Etot - E1)}{\partial E1}_{N,V,E} = 0$$

Equation A1.14 – Rearrangement of 1.8 (Frenkel and Smit 2002).

$$\frac{\partial ln\Omega 1(E1)}{\partial E1}_{N1,V1} = \frac{\partial ln\Omega 2(E2)}{\partial E2}_{N2,V2}$$

Equation A1.15 – Shorthand notation (Frenkel and Smit 2002).

$$\beta(E,V,N) \equiv \frac{\partial ln\Omega(E,V,N)}{\partial E}_{N,V}$$

Equation A1.16 – Eq 1.9 Rewritten with shorthand from 1.10 (Frenkel and Smit 2002).

$$\beta(E1,V1,N1) = \beta(E2,V2,N2)$$

To relate these formulas to the natural world, the maximum value of 1.8 represents the thermal equilibrium between two weakly interacting systems. If one were to place all of the energy in one system, the energy would transfer to the energy-less system until they were equal. At this point, ln Ω of the system is maximized. This implies a relationship between the number of eigenstates corresponding to a given energy and the thermodynamic entropy S of the system. This is agreeable with the second law of thermodynamics, which states entropy of a system is maximized when the system is at thermal equilibrium. We will therefore define entropy to be proportional to ln Ω (Frenkel and Smit 2002).

Equation A1.17 – Our definition of entropy, k_β is Boltzmann's constant (Frenkel and Smit 2002).

$$S(N,V,E) = k_\beta ln\Omega(N,V,E)$$

Recall that two systems are in equilibrium when β1=β2. This phenomenon can be observed in everyday life. The processes of two bodies at different temperatures brought into contact with each other reach thermodynamic equilibrium when they have the same temperature. Our intuition tells us that β must be related to absolute temperature. The thermodynamic definition of temperature is

Equation A1.18 – Thermodynamic definition of temperature (Frenkel and Smit 2002).

$$\frac{1}{T} = \frac{\partial S}{\partial E}_{V,N}$$

Equation A1.19 – Equation 1.10 solved for β (Frenkel and Smit 2002).

$$\beta = \frac{1}{k_\beta T}$$

This equation is known as the thermodynamic beta, and it is central to the functioning of replica exchange.

Appendix 2 – Pseudocode

A2.1 – DO_MD

```
void DO_MD ( ) {
    FOR step FROM 0 to NUMBER_OF_STEPS {
        IF ( step == 0 ) {
            IF    ( USE_REDS ) {
                INIT_REPLICA_EXCHANGE_REDS ( );
            } ELSE {
                INIT_REPLICA_EXCHANGE ( );
            }
        }
        …….
        IF ( USE_REDS ) {
            real currentEpot = enerd->term[F_EPOT];
            oldZeta = GET_ZETA(state);
            newZeta = UPDATE_ZETA(repl_ex, oldZeta, currentEpot);
            SET_ZETA(state, newZeta);
        }
        ……
```

```
        INTEGRATE_EQUATIONS_OF_MOTION ( );
        IF    ( USE_REDS ) {
            SCALE_FORCES ( force_vector , repl_ex , getZeta (state ) )
        }
        UPDATE_POSITIONS ( );
        IF ( step % EXCHANGE_INTERVAL == 0 ) {
            IF    ( USE_REDS ) {
                REPLICA_EXCHANGE_REDS ( );
            } ELSE {
                REPLICA_EXCHANGE ( );
            }
        }
    }
}
```

A2.2 – INIT_REPLICA_EXCHANGE_REDS

```
void INIT_REPLICA_EXCHANGE_REDS ( ){

      snew( re ->zetas, re->nrepl )                ;     allocate the 1D array of zetas

      snew( re ->biasCoefficients, re->nrepl )     ;     allocate the 1D array of pointers
                                                   ;     length is number of replicas

      FOR REPLICA      i      FROM 0...N-1  {

            snew( re ->biasCoefficients[ i ], 3 )  ;     allocate the number of columns
                                                   ;     to length 3, the three coefficients
      }
      re->biasCoefficients[ re->repl ][ 0 ] = re -> a11 ;  re->repl is the replica id

      re->biasCoefficients[ re->repl ][ 1 ] = re -> b11 ;  a11…c11 are parameters

      re->biasCoefficients[ re->repl ][ 2 ] = re -> c11 ;  each replica has a parameter file
}
```

For example, if this were replica 2 and re->a11 …c11 were provided 5, 6, 7 in the parameter file, at the end of this method this is what this process would consider the biasing coefficients and zeta arrays to look like.

Biasing Coefficients

	a11	b11	c11
repl 0	null	null	null
repl 1	5	6	7
…	null	null	null
…	null	null	null
repl N	null	null	null

Zetas

repl 0	repl 1	…	repl N
null	a value	null	null

A2.3 – UPDATE_ZETA

```
real UPDATE_ZETA ( replica_object   re,   real  oldZeta,   real energyPotential ) {
      real [ ] betas  = re -> betas;
      real  myBeta,  betaLeft,  betaRight = 0.0;
      int    index = re → repl;
      IF (index MOD 2 EQUALS 0 || index EQUALS re →nrepl – 1 || index EQUALS 0)
            betaLeft = betaRight = myBeta = betas[ index ];
      ELSE      {
                  betaRight =     betas[ index ];
                  betaLeft =      betas[ index ];
                  myBeta =        betas[ index ];
      }
      real MOVE_SIZE = 0.02;
      real NORMALIZING_FACTOR = 0.5;
      real zetaDiff = MOVE_SIZE * ( RANDOM_0_1 – NORMALIZING_FACTOR );
      real newZeta = oldZeta + zetaDiff;
      newZeta = BOUNDARY_CONDITIONS ( newZeta ) ;  keep zeta between 0 and 1
      real delta_1 = ((newZeta – oldZeta)*(betaRight – betaLeft)*
                              (energyPotential) ) / myBeta;
      real delta_2 = BIAS_EQUATION(re2, index, newZeta) –
                              BIAS_EQUATION(re2, index, oldZeta);
      real deltaE = delta_1 + delta_2;
```

```
IF ( deltaE <= 0 ) {
        re -> zetas [ re ->repl ]  = newZeta;              ;      Normal Keep
} ELSE IF ( METROPOLIS_CRITERION ( delta ) ) {
        re -> zetas [ re ->repl ]  = newZeta;              ;      Metropolis Keep
} ELSE {
        newZeta = oldZeta;                                 ;      Reject
}
          ;;;;;;;;;;;;;;;          End of Monte-Carlo Walker          ;;;;;;;;;;;;;;;
                    ;;;;;;;;;;;;;;;          Begin Output          ;;;;;;;;;;;;;;;
IF (NO OUTPUT FILE EXISTS)
        FILE = CREATE_FILE ( );


real averageForce, left_border, right_border = 0.0;
averageForce = ( -1 ) * energyPotential * ( ( betaRight - betaLeft ) / myBeta );
left_border = ( -1 ) * leftReplicaPotential * ( ( betaRight - betaLeft ) / myBeta );
right_border = ( -1 ) * rightReplicaPotential * ( ( betaRight - betaLeft ) / myBeta );


WRITE_TO_FILE (FILE, newZeta, averageForce, left_border , right_border);
}
```

A2.4 – SCALE_FORCES

```
VOID SCALE_FORCES ( rvec * VECTOR_OF_ATOMS,        replica_object  re )
      //      VECTOR_OF_ATOMS [ N ] [ 3 ]  ;
      //      Rows : Every Atom  Columns : x, y, z components of  Force
      real  myBeta,  betaLeft,  betaRight = 0.0;
      int index = re-> repl;
      real betas [ ] = re ->betas;                    // Initialized externally
      IF (index MOD 2 EQUALS 0 || index EQUALS NUM_REPLICAS – 1 || index
EQUALS 0)
            betaLeft = betaRight = myBeta = betas[ index ];
      ELSE      {
                  betaRight =     betas[ index + 1 ];
                  betaLeft =      betas[ index – 1 ];
                  myBeta =        betas[ index ];
      }
      real SCALING_FACTOR = ( ( 1 – re-> zeta ) * betaLeft + ( re-> zeta * betaRight ) )
/ myBeta ;
      FOR x from 0 to VECTOR_OF_ATOMS . length – 1{
            FOR y from 0 to 2 {
                  VECTOR_OF_ATOMS [ x ] [ y ] *= SCALING_FACTOR;
            }
      }     return;
}
```

A2.5 – TEST_FOR_REPLICA_EXCHANGE_REDS

```
VOID    TEST_FOR_REPLICA_EXCHANGE( replica_object re , real fresh_zeta ) {

    real [ ]  zetas = re->zetas ;
    int ** biasCoefficients = re-> biasCoefficents;
    boolean* bExchanged = re -> bEx;

            ;;;;;;;;;;;;;;          MPI Part of Method          ;;;;;;;;;;;;;;

    FOR REPLICA     i     FROM 0...NUM_REPLICAS – 1  {
        zetas [ i ] = 0.0;
    }
    zetas[ re -> repl ] = fresh_zeta;
    gmx_sum_sim (NUM_REPLICAS ,  re->zetas );
    IF ( FIRST CALL TO THIS METHOD ) {
        FOR REPLICA     i     FROM 0... NUM_REPLICAS – 1  {
            gmx_sum_sim ( 3 , re->biasCoefficients[ i ]  );
        }
    }
            ;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;    Continued…
```

```
    ;;;;;;;;;;;;;;;          Determine Exchange Part of Method          ;;;;;;;;;;;;;;;
  int  m = ( step / re -> nst ) % 2;
  real delta = 0.0;
  int a, b = 0;
  FOR REPLICA      index      FROM 1...N  {
      a = i – 1;
      b = i;
          IF (  index % 2 == m) {
              delta = CALCULATE_DELTA_REDS ( re , a , b)
              IF ( delta <= 0 ) {
                  bEx[ index ] = true;      ;      Normal Keep
              ELSE IF ( METROPOLIS_CRITERION ( delta ) )
                  bEx[ index ] = true;      ;      Metropolis Keep
              ELSE
                  bEx[ index ] = false;     ;      Reject
          }     ELSE {
                  bEx[ index ] = false;     ;      Auto Reject
          }
  }
}
                      ;;;;;;;;;;;;;;;          End          ;;;;;;;;;;;;;;;
```

MPI Mechanism Explained

At the beginning of every call to test_for_replica_Exchange_REDS, the array of zetas is overwritten with all zeroes, and the index of the process' replica is written with the value of the fresh_zeta. If there are four simulations, then the arrays of zetas look like below.

repl 0	repl 1	repl 2	repl 3
W	0	0	0

repl 0	repl 1	repl 2	repl 3
0	X	0	0

repl 0	repl 1	repl 2	repl 3
0	0	Y	0

\+

repl 0	repl 1	repl 2	repl 3
0	0	0	Z

---

repl 0	repl 1	repl 2	repl 3
W	X	Y	Z

The call to gmx_sum_sim will build a global array and place the sum of all the arrays into this global array. The global array is then redistributed to each process, overwriting the original arrays.

Order of Exchange Explained

Replica exchanges take place every so many steps. For example, consider if the replica interval is set to 100 steps. When the method is called, the step count will be an interval of 100, or step modulus 100 will equal 0. However, by dividing the step by the interval, and then modulus dividing by 2, we can determine the parity of the method call.

This occurs in this line of code.

```
int  m = ( step / re -> nst ) % 2;
```

Since the order of replicas is static, by taking the replica index modulus 2 and comparing it to the value of m, we create two disjoint sets of replicas which will attempt exchange each turn.

This occurs in this line of code:

```
IF (  index % 2 == m) {
```

A visual illustrating this process is shown below.

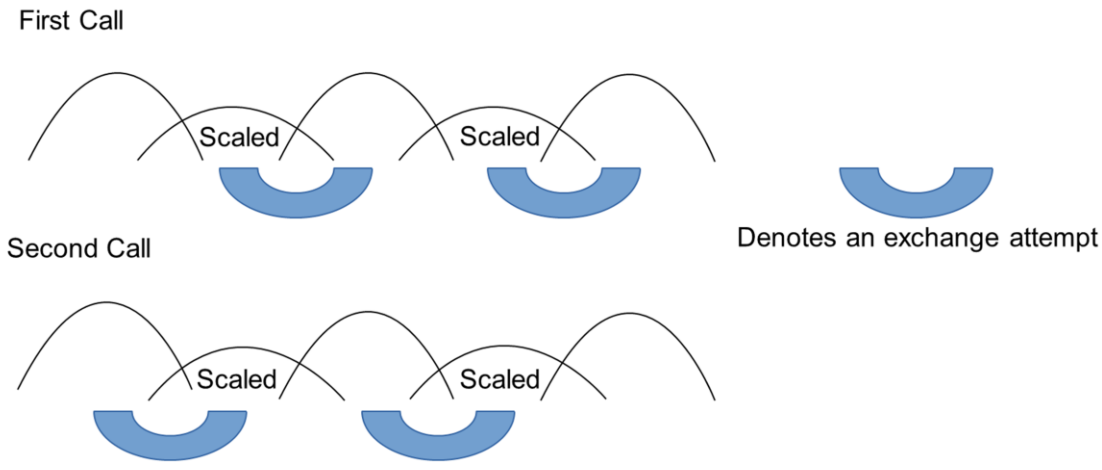


Figure 9, Patterns of Exchange

The blue oblong structures represent an attempted exchange between the two curves they connect. This pair of exchange patterns are alternated each time the TEST_FOR_REPLICA_EXCHANGE_REDS is called.

A2.6 – CALC_DELTA_REDS

```
REAL     CALC_DELTA_REDS ( replica_object       re , int     a, int       b )
      real part_1, part_2, part_3, part_4 = 0.0;
      real [ ] betas  = re -> betas;
      real A_myBeta, A_betaLeft, A_betaRight = 0.0;
      real B_myBeta, B_betaLeft, B_betaRight = 0.0;
      real A_energy = re -> Epot[ a ];
      real B_energy = re -> Epot[ a ];
      real A_zeta = re -> zetas[ a ];
      real B_zeta = re -> zetas[ b ];

      IF ( a MOD 2 EQUALS 0 || a EQUALS 0 )
            A_betaLeft = A_betaRight = A_myBeta = betas[ a ];
      ELSE
                  A_betaRight =       betas[ a + 1 ];
                  A_betaLeft =   betas[ a - 1 ];
                  A_myBeta =    betas[ a ];

      IF ( b MOD 2 EQUALS 0  | |  b = = NUM_REPLICAS – 1 || b EQUALS 0 )
            B_betaLeft = B_betaRight = B_myBeta = betas[ b ];
      ELSE
                  B_betaRight = betas[ i + 1 ];
```

```
                B_betaLeft   = betas[ i - 1 ];
                B_myBeta   =  betas[ i ];
        part_1 = – ( ( B_betaLeft – A_betaLeft ) * (B_zeta*B_energy –
                        A_zeta*A_energy ) );
        part_2 = – ( ( B_betaRight – A_betaRight ) * ( ( 1 – B_zeta ) * B_energy –
                        ( 1 – A_zeta ) * A_energy ) );
        part_3 = – B_myBeta * (BIAS_EQUATION ( re, b, B_zeta ) –
                        BIAS_EQUATION ( re, b, A_zeta ) );
        part_4 = – A_myBeta * (BIAS_EQUATION ( re, a, A_zeta ) –
                        BIAS_EQUATION ( re, a, B_zeta ) );

        real delta =    part_1 + part_2 + part_3 + part_4;
        return delta;
}

real BIAS_EQUATION ( replica_object         re,   int   index,     real zeta) {

        real        a = re -> biasCoefficients[ index ][ 0 ];
        real        b = re -> biasCoefficients[ index ][ 1 ];
        real        c = re -> biasCoefficients[ index ][ 2 ];
        return      a*zeta     +     b*(zeta^2)          + c*(zeta^3);
}
```

Appendix 3 – README to Download, Build, and Tour Code

```
/*********************************************|
//   Open a terminal.                         |
//                                            |
//   On a mac, this can be accomplished by    |
//   navigating mouse to search icon          |
//   in top right corner and clicking         |
//   then searching for "terminal"            |
//                                            |
//   Lines with $$$$ in front should          |
//   be copied and pasted onto the terminal   |
/*********************************************/

1)  To Download Our Gromacs

    _____________________________________

        Prereq: git
        To check if you have git:

        $$$$    which git

       This will either print its location "usr/bin/git" or print
       nothing
        If it prints nothing it isn't installed.

        To install git:

        $$$$    sudo apt install git

        type in your password
    _____________________________________


    First navigate the terminal to where you want to install gromacs
    Once there :

    $$$$    git clone -b REDS1_alt3

    PLEASE REQUEST FROM - csumma@uno.edu or go2432@wayne.edu


    Then type in the below password (Case Sensitive) :

    PLEASE REQUEST FROM - csumma@uno.edu or go2432@wayne.edu


    You now have all the code.

    To navigate to mdrun, where 99% of the changes are
```

$$$$ cd ./gromacs/src/programs/mdrun

$$$$ ls

The following files are contained in mdrun:

repl_ex.cpp - the replica exchange file

md.cpp - the molecular dynamics file

... all others in this directory are administrative files, i.e. set up the program to run, read command line options, ect.

---

2) A tour of the two files, md.cpp and repl_ex.cpp

Prereq: A text editor that can display/jump to line number X

To check if you have vim:

$$$$ which vim

This will either print its location "usr/bin/vim" or print nothing
If it prints nothing it isn't installed.

To install vim:

$$$$ sudo apt install vim

type in your password

To configure vim to display line numbers:

$$$$ echo set number >> ~/.vimrc

2a)

---

md.cpp

Prereq:

To open file:

$$$$ vi md.cpp

Now that we are in vim, to issue commands

You must first type either a ":" or a "/"

To jump to a line number:

:XXX

To search for a string:

/a string

More prereqs:

As of right now, we have been using Leap-Frog, which uses averaged half step kinetic energies to determine temperature We do not use verlocity verlet. So upon arriving at any if statements of the form

```
EI_VV(ir->eI) { ...
EI VV AK(ir->eI) { ...
```

the contents can be ignored as well as any comments about VV

Point in program where the most significant method of the mdrun program is called, do_md
Note that all the points following are contained in this method

:298

Point in program where Replica Exchange is initialized ...
primarily an administrative method call just to set up for later.

:645

Point in program where MD-Loop begins

:873

Point in program where step MD-Loop is on is compared to exchange step interval

:1043

If on a previous MD step, for example 100, an exchange occured, determined on :1908,
The globals are calculated for the quoted reason

"We need the kinetic energy at minus the half step for determining

the full step kinetic energy and possibly for T-coupling.
This may not be quite working correctly yet . . . . "

:1153

Point in program where trajectories at time, t, and positions, xyz_vec, are written (outputted) before updating

:1443

Point in program where energies and forces corresponding to the time t are obtained

:1462

Point in program where forces are scaled - happens every step
I may move this, however.

:1611

Point in program where coordinates are updated

:1615

Point in program where zeta is walked on by monte carlo - happens every step

:1897

Point in program where replica exchange is attempted - only every so often. boolean calculated on line 1043

:1908

2b)

---

repl_ex.cpp

Prereq: There are three ways md.cpp, the above file, uses this file

On line 645 of md.cpp

init_replica_exchange_REDS

:180

On line 1897 of md.cpp
updateZeta

:1092

On line 1908 of md.cpp
replica_exchange_REDS

:2263

See attached flow chart for a visual, (Figure 4), presentation of these three method calls

---

The replica exchange struct containing key values
Note: the number of these created == number of replicas

:92

The point where the replica exchange struct, line 92, is created and a couple values are filled in.
Note: most useful variables don't have value until test_for_replica_exchange, line 1557, is called

:178

The point where updateZeta is defined

:1087

The point, in test_4_RE_REDS, where all replicas receive a true or false value regarding exchange on this step.
Note: only even or odd replicas are even given a chance at a true value, alternating per call.

For example, on an even step

re0 re1 re2 re3

POSSIBLY TRUE FALSE POSSIBLY TRUE

On an odd step

re0 re1 re2 re3

FALSE POSSIBLY TRUE FALSE

:1557

The point in the test_for_replica_exchange method where calc_delta_REDS is called

:1762

The calc delta REDS method contents
Note that the last equation from slide 1, in swaps.pdf, is broken up into 4 parts which are summed
Also note we are conducting temperature re, so the ereTEMP switch case is all that need be concerned in this method

:1241

The point where the scaleForces method is defined

:1391

---

2) To install our gromacs
UBUNTU

Prereq: cmake
To check if you have cmake:

$$$$ which cmake

This will either print its location "usr/bin/cmake" or print nothing
If it prints nothing it isn't installed.

To install cmake:

$$$$ sudo apt install cmake

type in your password

Prereq: g++, or a c++ compiler
To check if you have g++:

$$$$ which g++

This will either print its location "usr/bin/ g++" or print nothing
If it prints nothing it isn't installed.

To install g++:

$$$$ sudo apt install g++

type in your password

Prereq: openMPI

To install g++:

$$$$ sudo apt-get install openmpi-bin openmpi-common openssh-client openssh-server libopenmpi1.3 libopenmpi-dbg libopenmpi-dev

type in your password

Step 1: Navigate to the gromacs folder

$$$$ cd ../../..

Step 2: Call Build Script

If you have GPU-Accleration

$$$$ ./BUILD_GROMACS_w_GPU.sh

If not

$$$$ ./BUILD_GROMACS.sh

Appendix 4 – README to Run GROMACS-REDS

```
/*********************************************|
//   Open a terminal.                         |
//                                            |
//   On a mac, this can be accomplished by    |
//   navigating mouse to search icon          |
//   in top right corner and clicking         |
//   then searching for "terminal"            |
//                                            |
//   Lines with $$$$ in front should          |
//   be copied and pasted onto the terminal   |
/*********************************************/

1)  To Download Our Scripting Environment

    _____________________________________

        Prereq: git
        To check if you have git:

        $$$$    which git

        This will either print its location "usr/bin/git" or print
        nothing
        If it prints nothing it isn't installed.

        To install git:

        $$$$    sudo apt install git

        type in your password
    _____________________________________


    First navigate the terminal to where you want to download the
    scripts
    Once there :

    $$$$    git clone

    PLEASE REQUEST FROM - csumma@uno.edu or go2432@wayne.edu


    Then type in the below password (Case Sensitive) :

    PLEASE REQUEST FROM - csumma@uno.edu or go2432@wayne.edu
```

You now have all the scripts along with a ready to simulate molecule, ala.pdb .

To begin a REDS simulation, navigate to automate_REDS

```
$$$$    cd ./reds/automate_REDS
```

Use the setup_reds script

```
/*****************************************************************\
//                                                                |
//   Usage: setup_reds.pl                                         |
//                                                                |
//                                                                |
//      Arguments:                                                |
//              pdbname         ; The name of the .pdb file       |
//              lowTemp         ; The lower temperature bound     |
//              highTemp        ; The upper temperature bound     |
//              replica_skip    ; The number of replicas to skip  |
//              seed            ; A random number generator seed  |
//                                                                |
//                                                                |
*****************************************************************/
```

An example:

```
$$$$ ./setup_reds.pl ala.pdb 300 600 3 1234
```

To begin a RE simulation, navigate to automate_RE

```
$$$$    cd ./reds/automate_RE
```

Use the setup_replica_exchange script

```
/*****************************************************************\
//                                                                |
//   Usage: setup_replica_exchange.pl                             |
//                                                                |
//                                                                |
//      Arguments:                                                |
//                              ; Only ala.pdb is currently supported |
//              lowTemp         ; The lower temperature bound     |
//              highTemp        ; The upper temperature bound     |
//              seed            ; A random number generator seed  |
//                                                                |
//                                                                |
*****************************************************************/
```

An example:

```
$$$$ ./setup_replica_exchange.pl 300 600 1234
```

When you have finished with a simulation and wish to run another one, to delete all the files you generated

```
$$$$    ./cleanup.sh
```

References

1. Abraham, M.J., et al., GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers. SoftwareX, 2015. 1: p. 19-25.
2. Alder, B.J. and T.E. Wainwright, Studies in molecular dynamics. I. General method. The Journal of Chemical Physics, 1959. 31(2): p. 459-466.
3. Arnol, V., d, Proof of a theorem of AN Kolmogorov on the preservation of conditionally periodic motions under a small perturbation of the Hamiltonian, Uspehi Mat. Nauk, 1963. 18: p. 13-40.
4. B. A. Berg and T. Neuhaus, Phys. Rev. Lett. 68, 9 (1992)
5. Bhatt, Manish & Rick, Steven. (2015). Random Walk Based Scaling for Optimizing Replica Exchange Molecular Dynamics.
6. Boltzmann, L., Vorlesungen über gastheorie. Vol. 1. 1896: JA Barth.
7. Case, D. and M. Karplus, Dynamics of ligand binding to heme proteins. Journal of molecular biology, 1979. 132(3): p. 343-368.
8. Denschlag, R., M. Lingenheil, and P. Tavan, Efficiency reduction and pseudo-convergence in replica exchange sampling of peptide folding–unfolding equilibria. Chemical Physics Letters, 2008. 458(1-3): p. 244-248.
9. Earl, D.J. and M.W. Deem, Parallel tempering: theory, applications, and new perspectives. Phys Chem Chem Phys, 2005. 7(23): p. 3910-6.
10. eigenvector | Definition of eigenvector in US English by Oxford Dictionaries. Oxford Dictionaries | English; Available from: .
11. Feig, M., Computational protein structure refinement: almost there, yet still so far to go. Wiley Interdisciplinary Reviews: Computational Molecular Science, 2017. 7(3).
12. Fermi, E., et al., Los Alamos Report No. LA-1940, 1955. 978.
13. Frenkel, D. and B. Smit, Understanding molecular simulation : from algorithms to applications. 2nd ed. ed. 2002, San Diego ; London: Academic Press.

14. Gibbs, J.W., Elementary principles in statistical mechanics. 1902, New York: Charles Scribners Sons.
15. Kolmogorov, A. On the conservation of conditionally periodic motions under small perturbation of the Hamiltonian. in Dokl. Akad. Nauk. SSR. 1954.
16. Lee, A.J. and S.W. Rick, Improving replica exchange using driven scaling. J Chem Phys, 2009. 131(17): p. 174113.
17. Levitt, M. and R. Sharon, Accurate simulation of protein dynamics in solution. Proceedings of the National Academy of Sciences, 1988. 85(20): p. 7557-7561.
18. Levitt, M. and A. Warshel, Computer simulation of protein folding. Nature, 1975. 253(5494): p. 694.
19. Machta, J., Strengths and weaknesses of parallel tempering. Physical Review E, 2009. 80(5): p. 056706.
20. McClure, David. "DEGENERACY AND ALL THAT." Handout from Statistical Mechanics – PH 664. Portland State University. 2/25/2010.
21. McGraw-Hill dictionary of scientific and technical terms. 6th ed. 2003, New York: McGraw-Hill. xvii, 2380 p.
22. McQuarrie, D.A., Statistical thermodynamics. 1973, Mill Valley, Calif.: University Science Books. xi, 343 p.
23. Merzbacher, E., Quantum mechanics. 2nd ed. ed. 1970, New York ; London: Wiley.
24. Message P Forum. 1994. Mpi: a Message-Passing Interface Standard. Technical Report. University of Tennessee, Knoxville, TN, USA.
25. Monod, J., J. Wyman, and J.-P. Changeux, On the nature of allosteric transitions: a plausible model, in Selected Papers in Molecular Biology by Jacques Monod. 1978, Elsevier. p. 593-623.
26. Möser, J., On invariant curves of area-preserving mappings of an annulus. Nachr. Akad. Wiss. Göttingen, II, 1962: p. 1-20.
27. Nave, C.R. The Hamiltonian. 2005; Available from: .

28. Nymeyer, H., How efficient is replica exchange molecular dynamics? An analytic approach. Journal of chemical theory and computation, 2008. 4(4): p. 626-636.
29. Ostermeir, K. and M. Zacharias, Advanced replica-exchange sampling to study the flexibility and plasticity of peptides and proteins. Biochimica et Biophysica Acta (BBA)-Proteins and Proteomics, 2013. 1834(5): p. 847-853.
30. Patrascioiu, A., The ergodic-hypothesis: a complicated problem in Mathematics and physics. Los Alamos Science, 1987. 15: p. 263-279.
31. Perutz, M. Structure of hemoglobin. in Brookhaven Symp Biol. 1960.
32. Perutz, M.F., et al., Structure of haemoglobin: a three-dimensional Fourier synthesis at 5.5-Å. resolution, obtained by X-ray analysis. Nature, 1960. 185(4711): p. 416.
33. Rahman, A., Correlations in the motion of atoms in liquid argon. Physical Review, 1964. 136(2A): p. A405.
34. Rick, S.W., Replica exchange with dynamical scaling. J Chem Phys, 2007. 126(5): p. 054102.
35. Summa, C.M. and M. Levitt, Near-native structure refinement using in vacuo energy minimization. Proceedings of the National Academy of Sciences, 2007. 104(9): p. 3177-3182.
36. G. M. Torrie and J. P. Valleau, J. Comput. Phys. 23, 187 (1977)
37. Wikipedia contributors. (2018, April 10). Degenerate energy levels. In a Wikipedia, The Free Encyclopedia. Retrieved 22:54, April 16, 2018, from a https://en.wikipedia.org/w/index.php?title=Degenerate_energy_levels&oldid=835765095
38. Zuckerman, D.M. and E. Lyman, A second look at canonical sampling of biomolecules using replica exchange simulation. Journal of chemical theory and computation, 2006. 2(4): p. 1200-1202.