UNIVERSITY OF CALIFORNIA SAN DIEGO

Effective Utilization of Next Generation Hardware for Complex Molecular Dynamics Simulations

A dissertation submitted in partial satisfaction of the

requirements for the degree Doctor of Philosophy

in

Chemistry

by

Charles Lin

Committee in Charge:

Professor Ross C. Walker, Chair
Professor Rommie E. Amaro, Co-Chair
Professor Ruben Abagyan
Professor Patricia Jennings
Professor Elizabeth Komives

2018

The Dissertation of Charles Lin is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____
Co-Chair

_____
Chair

University of California San Diego

2018

Dedication

This thesis is dedicated to my parents, Rongjau and Chin, and sister, Amanda, and my dog, Midnight, who have been extra supportive throughout these last four years.

Epigraph

"A journey will have pain and failure. It is not only the steps forward that we must accept. It is the stumbles. The trials. The knowledge that we will fail. That we hurt those around us.

But if we stop, if we accept the person we are when we fall, the journey ends."

Brandon Sanderson, *Oathbringer*, 2017

Table of Contents

List of Abbreviations

Amber: Amber Molecular Dynamics Package Suite

ASIC: Application-Specific Integrated Circuit

CPU: Central Processing Unit

DP: Double Precision

FEP: Free Energy Perturbation

FFT: Fast Fourier Transform

Flop: floating point operation per second

FP: Fixed Precision

FPGA: Field-Programmable Gate Array

GB: Generalized Born

GPU: Graphics Processing Unit

MD: Molecular Dynamics

NPT: Isothermal-isobaric ensemble with number of particles, pressure, and temperature held constant

NVE: Microcanonical ensemble with number of particles, volume, and energy held constant

NVT: Canonical ensemble with number of particles, volume, and temperature held constant

PBC: Periodic Boundary Conditions

PME: Particle Mesh Ewald

SP: Single Precision

TI: Thermodynamic Interation

List of Figures

List of Tables

Preface

It is an exciting time in technology. As ideas that have long been theorized have become actualized and once long heralded standards are destroyed, the interfaces between different fields have meshed together, and perception of what is virtual and tangible have begun to meld together. In just the past few years, virtual reality and augmented reality have emerged from pure fantasy to reality. Large technology companies such as Facebook and Google have shifted their focus to these fields focusing purely on enhancing their camera experiences. Quantum computing has begun to make a forefront in technology as D-Wave [1] launched their quantum annealing machines soon to be disrupting the cryptology field. Now Intel and Google have announced their own quantum machines offering up to 76 qubit calculations. While, this technology is still in its infancy, these machines can potentially easily break RSA encryption, a method involving finding a 2 prime number keys to a large number, due to its ability to make guesses on many orders of magnitude faster than today's fastest machines. As of the RSA conference in 2017, cryptologists have yet to come up with stronger encryption methods. Virtual currencies such as bitcoin [2] and ethereum [3] have made 10,000% returns in the last five years. These involve solving hashes in blockchains by solving simple but numerically expensive mathematical equations and adding them to a shared network. And lastly, with stronger computing power and advances in neuroscience, deep learning has taken the entire computer science field by storm powering everything from image recognition to natural language processing. In short, the landscape of technology has massively changed in just the four years since I began this Ph. D.

However, outside of quantum computing, each of these advances is still powered by incremental advances of hardware that have existed for decades, primarily computers, more

specifically the processing powers of the central processing unit (CPU) and graphical processing unit (GPU). At the forefront of these fields stand Intel and AMD for their advancements in CPUs and NVIDIA and AMD for GPUs. As GPUs have become the gold standard in today's world for computing, NVIDIA's stock price has grown over 1000%. However, it was not always this way. Graphics cards were initially built for graphics processing in games, and have been a hallmark in that industry. As games required more complex operations such as calculation of shaders, their ability to do scientific calculation became more feasible, and thus the CUDA programming language was created. In 2012, the first major applications in scientific computing using graphics cards were in the life sciences field. This came in the form of molecular dynamics, which instantly sped up calculations by over 50x a single thread[4, 5]. On the CPU side, Intel released a new processor line called the Xeon Phi processor that aims to mimic GPUs but with higher processing power and fidelity, making it a potentially powerful solution for even the most complex scientific applications.

With the many fields that have birthed from CPU and GPU technologies that are fundamental to the vital operations of nearly all industries, it is vital that life sciences is not left behind as a relic. For what drives innovation may have once been science, but the market has in the last few years shifted to the everyday consumer. The market for graphics cards is for gamers and big technology companies, the market for processors is home computers, and a large portion of today's silicon is being built towards power efficient ARM CPUs in the ever demanding smartphone market. Scientific computing merely takes in the advances of consumer technology and incorporates it into its own, but if no one is willing to help it expand, it will be left behind.

Acknowledgements

First and foremost I would like to thank my PhD advisor, Ross Walker, for his guidance and support through this entire process which not only included academia and my PhD, but also opportunities and networking in the pharmaceutical and tech industry.

To my doctoral committee, Rommie Amaro, Elizabeth Komives, Ruben Abagyan, and Patricia Jennings for their support throughout my adventurous PhD career.

I would like to thank members who were part of the Walker Lab at SDSC, Dr. Benjamin Madej, Dr. Andreas Goetz, Dr. Aage Skjevik, Dr. Daniel Mermelstein, Dr. Perri Needham, and Allan Yeh for their guidance and discussions throughout this process.

I would like to thank the following people from GSK, where I have spent the last 18 months completing my PhD, with useful conversations and support: Dr. Alan Graves, Dr. Guanglei Cui, Dr. Nikolay Plotnikov, Dr. Neysa Nevins, Dr. Jamel Meslamani, Dr. Eric Arnoult, Craig Kapfer, Gordon Madise, and Charitha Belluru Shetty.

I would like to thank the following members from Intel for the immense amount of time, effort, and support in the last two years, this work would definitely not have been possible or feasible without the immense time commitment they have given towards the project: Dr. Ashraf Bhuiyan, Dr. Tareq Malas, Dr. Dayle Smith, Ram Ramanujam, Dr. Roland Shultz, and Dr. Mike Brown.

I would like to thank Dr. Mike Gilson, Ido Ben-Shalom, Dr. Callum Dickson, Dr. Knut Teigen, Scott Resnick, Dennis Tseng, Travis Hance, Victor Pan, Rob Alberstein, Bai Nitin, Amit

Harode, Benny Mathews for their countless hours of help with coding, debugging, algorithmic design, and development on the projects in this thesis.

I would like to thank the following members of the Amber community for miscellaneous help and support with merges, keeping me updated with major code changes, and other miscellaneous aspects: Dr. David Case, Dr. Dave Cerutti, Dr. Jason Swails, Dr. Adrian Roitberg, and Dr. Darrin York.

I would also like to thank my former research advisors from my undergraduate career, Dr. Chenlong Li and Dr. Shou-Mei Wu who helped encourage this path. I would like to thank lab members from my former labs: Dr. Ryan Pavlowski, Dr. Guqin Shi, Dr.Chung-An Chen, and Eileen Tran.

I would like to thank Linda Shen and William Li for converting my pencil drawn figures into higher quality figures that are used in this thesis.

Lastly, I would like to thank my friends who did not contribute to my thesis, but who have encouraged me throughout this journey and told me not to give up: Randy Tsai, Gavin Bauman, Vania Chen, Bradley Lai, Susanna Lee, Alex Lambert, Victoria Cheung, Elizabeth Leung, Alan Su, Derek Ye, Eileen Tran, Diana Luu, Nancy Luu, Sharon Xu, Judy Pham, and Stephen Trac.

Chapter 3, in part is currently being prepared for submission for publication of the material. Lin, Charles; Malas, Tareq; Bhuiyan, Ashraf; Yeh, Allan; Walker, Ross C. The dissertation author was the primary investigator and author of this material.

Chapter 5, in part, is a reprint of the material as it appears in Simulation of lipid bilayer self-assembly using all-atom lipid force fields in Physical Chemistry Chemistry Physics 2016. Shjevik Age A.; Madej, Benjamin D.; Dickson, Callum. J.; Lin, Charles; Teigen, Knut; Walker, Ross, C., 2016. The dissertation author was an author of this paper.

VITA

2012	Summer Intern
	Shao-Mei Wu Lab, College of Pharmacy, Kaohsiung Medical University
2013-2014	Undergraduate Researcher
	Chenglong Li Lab, College of Pharmacy, Ohio State University
2014	Bachelor of Science, Pharmaceutical Science
	Ohio State University
2014-2017	Graduate Student Researcher, Walker Lab
	University of California, San Diego
2017-2018	Research Fellow
	GlaxoSmithKline
2018	Doctor of Philosophy in Chemistry
	University of California San Diego

PUBLICATIONS

Lin, C., Malas, T., Bhuiyan, A., Yeh, A., Walker, R. Extending Amber Molecular Dynamics CPU scaling to thousands of cores. *In prep.*

Lee, T., Mermelstein, D., Cerutti, D., Lin, C., LeGrand, S.,Giese, T.J., Roitberg, A., Case, D.A., Walker, R.C., York, D.M. GPU-accelerated molecular dynamics and free energy methods in AMBER 18: performance enhancements and new features. *In review.*

Mermelstein, D. J., Lin, C., Nelson, G., Kretsch, R., McCammon, J. A., Walker, R. C. Fast and Flexible GPU Accelerated Binding Free Energy Calculations within the AMBER Molecular Dynamics Package, *Journal of Computation Chemistry.* DOI: 10.1002/jcc.25187

Skjevik, A. A., Madej, B. D., Dickson, C. J., Lin, C., Teigen, K., Walker, R.C., Gould, I. R., Simulation of lipid bilayer self-assembly using all-atom lipid force fields, *Phys. Chem. Chem. Phys.*, 2016,18, 10573-10584

D.A. Case, S.R. Brozell, D.S. Cerutti, T.E. Cheatham, III, V.W.D. Cruzeiro, T.A. Darden, R.E. Duke, D. Ghoreishi, H. Gohlke, A.W. Goetz, D. Greene, R Harris, N. Homeyer, S. Izadi, A. Kovalenko, T.S. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, D.J. Mermelstein, K.M. Merz, Y. Miao, G. Monard, H. Nguyen, I. Omelyan, A. Onufriev, F. Pan, R. Qi, D.R. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, J. Shen, C.L. Simmerling, J. Smith, J. Swails, R.C. Walker, J. Wang, H. Wei, R.M. Wolf, X. Wu, L. Xiao, D.M. York and P.A. Kollman (2018), AMBER 2018, University of California, San Francisco.

Case, D. A.; Cerutti, D. S.; Cheatham III, T. E.; Darden, T. A.; Duke, R. E.; Giese, T. J.; Gohlke, H.; Goetz, A. W.; Greene, D.; Homeyer, N.; Izadi, S.; Kovalenko, A.; Lee, T. S.; LeGrand, S.; Li, P.; Lin, C.; Liu, J.; Luchko, T.; Luo, R.; Madej, B.; Mermelstein, D.; Merz, K. M.; Monard, G. H.; Nguyen, H.; Omelyan, I.; Onufriev, A.; Pan, F.; Qi, R.; Roe, D. R.; Roitberg, A.; Sagui, C.; Simmerling, C. L.; Botello-Smith, W. M.; Swails, J.; Walker, R. C.; Wang, J.; Wolf, R. M.; Wu, X.; Xiao, L.; York, D. M.; Kollman, P. A., AMBER 2017, University of California, San Francisco.

Case, D. A.; Cerutti, D. S.; Cheatham III, T. E.; Darden, T. A.; Duke, R. E.; Giese, T. J.; Gohlke, H.; Goetz, A. W.; Greene, D.; Homeyer, N.; Izadi, S.; Kovalenko, A.; Lee, T. S.; LeGrand, S.; Li, P.; Lin, C.; Liu, J.; Luchko, T.; Luo, R.; Madej, B.; Mermelstein, D.; Merz, K. M.; Monard, G. H.; Nguyen, H.; Omelyan, I.; Onufriev, A.; Pan, F.; Qi, R.; Roe, D. R.; Roitberg, A.; Sagui, C.; Simmerling, C. L.; Botello-Smith, W. M.; Swails, J.; Walker, R. C.; Wang, J.; Wolf, R. M.; Wu, X.; Xiao, L.; York, D. M.; Kollman, P. A., AMBER 2016, University of California, San Francisco.

ABSTRACT OF THE DISSERTATION

Effective Utilization of Next Generation Hardware for Complex Molecular Dynamics Simulations

By

Charles Lin

Doctor of Philosophy in Chemistry

University of California San Diego, 2018

Professor Ross C. Walker, Chair

Professor Rommie E. Amaro, Co-Chair

As technology moves forward with newer hardware, new computational techniques, and new algorithms, it is important for science to keep up with these advances. Without people maintaining code to the latest standards and modifying the underlying algorithms to closely match the shifting hardware landscape, scientific software easily becomes outdated and deprecated on the latest architectures. Similarly as technology advances and acapability increases so the underlying methods should be adapted, extended or replaced in order to fully

realize the scientific potential of the underlying improvements in the hardware. This research focuses on developing faster and higher accuracy free energy prediction methods as well as the development of simulation techniques necessary to improve the sampling efficiency of all atom, condensed phase, molecular dynamics simulations on exa-scale computational hardware. Specifically, it will cover efforts to develop GPU accelerated thermodynamic integration algorithms, low level code optimization on next generation CPU architectures, techniques for more efficient 3D domain decomposition and work to explore more efficient sampling techniques. These methods have been implemented as part of this work within the Amber software package. Additionally, to show the accuracy and performance of the code, comparisons against previous Amber versions as well as existing free energy prediction packages such as Schrodinger's FEP+ will be analyzed and discussed.

Chapter 1: What is Molecular Dynamics

Molecular dynamics [6] is the idea that it is possible to generate time based information given a set of initial coordinates, particularly in the space of atomistic simulations. The idea follows upon several ideas particularly ball and spring methods which follow Newton's second law of motion. One of the most interesting ideas upon molecular dynamics is why is Newton's second law of motion, F=ma, still a good approximation at the molecular level?

To answer this, it is important to look at an idea called de Broglie wavelengths [7]. Which is the idea that

$$\lambda = \frac{h}{\sqrt{2*\pi*m*k_b*T}} \qquad (1)$$

where lambda (λ) is the de Broglie wavelength, h is plank's constant $6.626*10^{-34}$ m^2 kg/s, m is the mass of the object, k_b is Boltzmann's constant $1.38*10^{-24}$ m^2 kg s^{-2} K^{-1}, and T is temperature. Using this equation it is possible to show that for most atoms at room temperature the de Broglie wavelength is significantly smaller than the distance between atoms. This implies that quantum interference between atoms does not occur and they can be treated with a classical model. Note this approximation only holds at temperatures several degrees or more above absolute zero and with atomic masses greater than 1. For example, when looking at hydrogen at 300K, this approximation starts to break down, as hydrogen has a one angstrom de Broglie wavelength. Molecular dynamics relies on the idea that the internal quantum interference of molecules is smaller than the distance between atoms. For most atoms, the de Broglie wavelength is significantly smaller than a bond length of around 1-2 angstroms, thus, they can be approximated using molecular dynamics. However, for hydrogen, molecular dynamics often constrains hydrogen atoms, so their quantum like behavior is suppressed.

Knowing that a Newtonian approach will mostly work for atomics systems, it is possible to determine the position of molecules using standard kinematic equations:

$$v_f = v_i + a * t \qquad (2)$$

$$x = v_i * t + \frac{a*t^2}{2} \qquad (3)$$

$$F = m * a \qquad (4)$$

By knowing the basic atom information it is possible to get the initial position and mass of the molecules. Using equation 2 it is possible to get the final velocity (v_f) y knowing the initial velocity (v_i) and the acceleration (a) and time step of the system (t). Using equation 3 its possible to get the new displacements of the system (x) by knowing the initial velocity, acceleration, and time. It is lastly possible to calculate the acceleration by knowing the force (f) and mass (m) of the system using equation 4. By setting initial velocities to a random Gaussian distribution, it is possible to solve all of the equations as long as it is possible to figure out the force on the molecules as a function of their position at a given moment in time.

Energy Equation

To solve for the force, it is important to remember that the gradient of the energy, ∇E, is the partial forces. For Amber, this energy equation is:

$$V(r^N) = \sum_{\text{bonds}} k_b(l - l_0)^2 + \sum_{\text{angles}} k_a(\theta - \theta_0)^2 + \sum_{\text{torsions}} \sum_{n} \frac{1}{2} V_n[1 + \cos(n\omega - \gamma)] + \sum_{j=1}^{N-1} \sum_{i=j+1}^{N} f_{ij} \left\{ \epsilon_{ij} \left[\left(\frac{r_{0ij}}{r_{ij}} \right)^{12} - 2 \left(\frac{r_{0ij}}{r_{ij}} \right)^{6} \right] + \frac{q_i q_j}{4\pi\epsilon_0 r_{ij}} \right\} \qquad (5)$$

Where k_b is a the bonds constant, l is the current angle being evaluated, l_0 is the equilibrium bond distance, k_a is the angles constant, θ is the current angle of the atoms, θ_0 is the equilibrium angle distance, V_n is the torsional amplitude for the torsional energy, ω is the psi angle of the torsion, γ

is the phi angle of the torsion, and lastly n controls the periodicity of the torsion. For the last portion of the equation r_{0ij} represents the equilibrium distance of a non bond interaction, while r_{ij} represents the current distance. f_{ij} and ε_{ij} are parameters for the non bond energies, q_i and q_j represent the charges for the pair of atoms, and lastly, ε_0 is the vacuum permeability. This equation may look complicated, but it's the sum of five different energies: bonded energies, angle energies, torsional (dihedral) energies, non-bonded energies, and charged energies. Note, that this equation is in internal coordinates, when in reality all the data is generally in 3D Cartesian coordinates.

The bond, angle, and torsional energies are derived from a ball and spring model, where the energy of a spring attached to two atoms is simply a constant multiplied by the displacement squared. This allows an optimal r_0 distance when the bond is in an ideal location, and a repulsion when the r is too close to r_0 and attraction when r is further away from r_0. This is similar to the derivation of the angle and dihedral energies where there is an attraction and repulsion term is if the angles are too acute and obtuse and whether the phi psi angles in a torsion are optimal. A simple harmonic model that is used here is suitable for the movement of large atoms across space, but it lacks the ability to keep track of the movement of electrons and therefore it is insufficient to show the creation and breakage of bonds in a system.

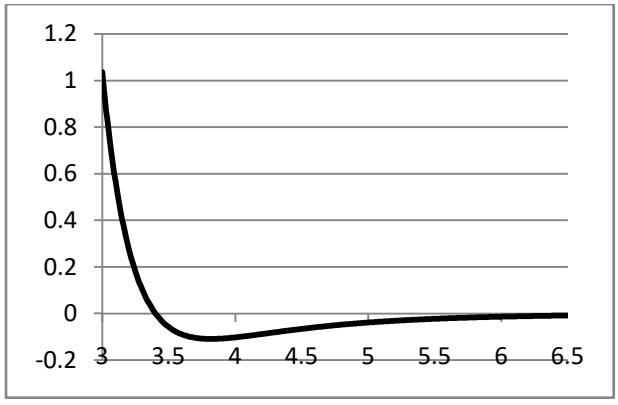


Figure 1: Shows van der waals forces for Argon atoms based on distance.

The non-bonded energies come from two sources: the Lennard Jones non-charge interactions based on atomic size and the charge based interactions. The non-charge basedinteractions are modeled from the interatomic potential energy called the Lennard-Jones potential [8], which approximates an interatomic potential. This curve has two segments near its optimal energy: a repulsive r^{12} term increases exponentially as atoms become too close together and an attractive r^6 term that fades off as atoms move further from each other (Figure 1). This gives an optimal distance for non-bonded atoms to be next to each other.

Lastly, there is a standard charge equation, which is similar to equation of two point charges interacting with each other. This term is the only long distance term that needs to be calculated as it decreases on a scale of 1/r, which means that the effects of the electrostatics is much further than any of the other terms. The bonds, angles, and dihedrals only matter for the atoms in those sets, while the Lennard Jones potential quickly approaches zero, as a pair of atoms get further away from each other in space.

The derivative of the energy expression with respect to position gives the force. By combining each of these different force terms together, it is possible to obtain the total force on an atom with a given position due to all other atoms. From the equations above we can see that each specific atom has many different components of forces acting upon it. As one increases the number of atoms in the system, the number of interactions increases on the order of N^2. Given a large enough simulation box with sufficient atoms included it is possible to start to model bulk properties.

The Amber Software Package

Although there are many different molecular dynamics software packages used widely today including but not limited to OpenMM [9], GROMACS[10-13], CHARMM [14, 15], NAMD[16], and LAMMPS [17], this thesis will primarily focus on a software package called Amber [18-20]. As of April 2018, the latest version of this software package is v18. The Amber software package is split into two portions: AmberTools and Amber. Where AmberTools is open source, free software that is released annually, Amber is a paid version that has limited functionality and released biennially.

Amber's free portion contains everything that is needed to set up, run, and analyze a simulation. It contains close to twenty different packages. For setup it contains software packages like *antechamber* [21] that help determine the charge states of a system, *reduce* (ref) that add the hydrogen to a protein structure, and *xleap*/*tleap* [22] which populate the parameters for a system and generate the necessary input files. For molecular dynamics engines, AmberTools offers *sander* which can do classical molecular dynamics (MD) and combined quantum mechanical/molecular mechanical (QM/MM) simulations as well as *mgdx [23]*, which offers experimental optimizations for classical MD. For post processing, AMBER offers *cpptraj* and *pytraj* [24], which help analyze trajectories generated from simulation providing functionality such as clustering of similar results and reducing frames of the trajectory. Lastly, in terms of parameterization, *parmed* [25] and *paramfit* [26] allow the creation of new parameters and the editing of parameters. Although these software packages are the ones bundled with AMBER, users are not limited to using just these software packages and are free to choose what they believe are best for them.

With the paid portion of AMBER, the user gains one major thing over the free version: speed. The biennial release version of AMBER includes *pmemd* and its derivatives. *Pmemd* is the fastest version of the molecular dynamics engine that AMBER offers that includes optimizations for CPU based simulations and GPU based simulations. For CPU simulations, optimizations are added for the Intel MIC architecture as well as the Intel Xeon and AMD Epyc architectures. For GPU simulations, support is added for NVIDIA GPUs via the CUDA architecture. These versions of the code perform orders of magnitude faster than their sander equivalent. On top of optimizations for architectures, this portion of the code based also includes optimizations such as 3-D domain decomposition and polarizable forcefield simulations *pmemd.gem [27]* and *pmemd.amoeba [28]*.

Outside of the underlying software suite, AMBER also refers to the open domain portion that contains the parameters needed to run these simulations termed forcefields.

Forcefields

In order to solve the energy equation in equation 5 above, several additional pieces of information are needed. In the simplest approximation, the positions and displacements between sets of atoms as well as the partial charge, VDW parameters, and weight are needed. Beyond that, constants (parameters) are needed for each term that defines the resting distances between sets of differing atoms as well as a constant that describes the magnitude of the restoring forces.

A set of parameters for a certain type of problem is called a forcefield. Although, there are general forcefields that describe specific atoms, sometimes these variables do not truly capture the physical properties of molecules. Therefore, it is usually important to parameterize for molecular sets of interest. This includes specific environments such as lipids and amino acids.

Examples of why generic atom based force fields cannot be used include things such as lipid density of a membrane could be too high or too low based on parameterization of the hydrophobic side chains, or the phi-psi angles in a ramachandran plot of an amino acid do not match what is determined experimentally.

In AMBER, forcefield family presently exist of: the all atom forcefield gaff [21, 29] and gaff2 which contain broad parameter sets describing biologically relevant small molecules, the protein forcefield FF14SB[30], which are targeted at proteins and contain parameterization for amino acids, and Lipid17, an expansion on Lipid14 [31, 32], contains forcefield parameters for lipids. These forcefields typically designed to work as one package since they share common approaches to parameterization. For example, mixing and matching different classes of Amber forcefields will typically yield better results than mixing differently classes of Amber and CHARMM forcefields. This is because forcefields are parameterized using specific conditions in which they run best. This may include having different cutoff schemes such as distance or cutoff type.

Implicit vs Explicit Solvent

Generally the computational complexity of molecular dynamics simulations in their most basic form scales on an order of $O(n^2)$ primarily because the nonbond and charge portion of the equation require evaluation of the interaction between all pairs of atoms. Fortunately, there are a number of several optimizations that are commonly used to decrease simulation time. The two most basic methods are Generalized Born (GB) [33] and Particle Mesh Ewald (PME) [34-36] methods.

Generalized Born is an implicit solvent water method. This means that water molecules are not explicitly present in the simulation but rather included by way of terms included in the energy equation. There are many different approaches to solving the Generalized Born equation, but in most cases it involves explicitly including all atoms of interest (usually proteins), and having them interact with all other atoms in the system. This creates an $O(n^2)$ simulation time as all atoms need to interact with all other atoms in the system, however, by removing water in the system it decreases the amount of time needed to run these simulations since the total atom count (n) is reduced. There is also a reduction in viscosity which can lead to an effective increase in the rate at which phase space is explored [37].

Particle Mesh Ewald (PME), which will be for the most part the primary focus of this thesis, is an explicit solvent method. This means that all solvent molecules, typically but not always water, have an atomic representation in the simulation. This creates, in a naïve implementation, a very expensive and dense simulation. It also creates a problem of what happens outside the simulation box. What happens if a water molecule needs to move out the edge of a box because the forces tell it to? To deal with this PME uses something called periodic boundary conditions (PBC) [38] which make it so a simulation unit is translated and copied infinitely across a simulation space like a tiled wall but in this case in three dimensions (Figure 2).

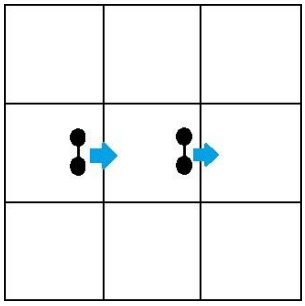


Figure 2: Periodic boundary conditions imitate the same home cell infinitely. When an atom leaves a cell, it comes back in from the opposite corner.

Another problem that arises with PME is what happens with the charge-charge interactions since they do not drop off as quickly as any other component of the calculation. Indeed the drop off is sufficiently slow that there can be an appreciable interaction even between the periodic images of the simulation box. To deal with this, a calculation, the namesake of this method is used called particle mesh ewald. The idea of PME is that a fine grid is created where the grid spacing is ideally, due to the way in which the effective accuracy of the approach is tuned, a one angstrom based grid, and each atom's position corresponds to one of these grid cubes. A charge map is created in which each position where there is an atom, its corresponding charge is placed in that grid cell. The grid charge is spread across neighboring cells to however far the user wishes to place that spread (most codes default this as up to 4 grid cells away). By calculating a Fast Fourier Transform (FFT) [39] on this grid it is then possible to calculate the effect of the reciprocal space (the space not owned by the simulation) on the real space atoms. By utilizing optimization in the PME code, the code goes from $O(n^2)$ to about $O(n*\ln(n))$ in time. Alternative to PME are methods such as Isotropic Periodic Sum[40], which creates artifacts of the home cell beyond a cutoff distance to simulate an infinite field, and reaction field[41], which simulates distances beyond a certain cutoff as a uniform dielectric.

General Optimizations for PME

Beyond using a particle mesh Wwald method the explicit solvent implementation requires several other optimizations to scale across multiple cores, processor sockets, and compute nodes that make up modern high performance computing systems. One of the most useful optimizations involves the nonbond calculations. This involves changing the Lennard Jones equation from being an all atom calculation to utilizing cut offs. Due to the Lennard Jones

calculations quick drop off towards zero relative to distance, codes usually only compute Lennard Jones interactions of atoms whose distances are within this cutoff. In Amber, the code follows the Lennard Jones up to the cutoff, and after the cutoff all Lennard Jones interactions outside of this range are disregarded.

A naïve implementation of this ends up becoming close to $O(n^2)$ still due to the need to do an $O(n^2)$ search of all atoms to find whether or not their partners are within the cutoff range. To optimize this, a neighbor list (termed a pairlist) is often used. The way this is usually done is by binning the atoms into bins slightly bigger than half the cutoff size. In this case, it is only necessary to search two bins outward from where an atom is for its pairs. The pairlist is expensive to build because it is essentially an $O(n^2)$ calculation, but it is faster than spending the time to check for pairs and calculating out the energy. The key thing that makes the pair list more efficient is that it is not necessary to rebuild the pairlist each step. In fact AMBER only rebuilds it approximately once every 15-25 steps. This is because the pair list is not actually built on atoms that are only the cutoff distance away, but a distance of the cutoff + a small skin value. This means that as long as atoms do not move beyond the cutoff distance plus the skin value, there is no need to rebuild the pairlist, but the moment an atom moves a further distance than the cutoff plus skin value, the entire pairlist needs to be rebuilt. By utilizing these optimizations, the nonbond calculation becomes a lot faster.

Shake [42] is the last major algorithm that is used for optimizations. Similar algorithms often used are the RATTLE [43] and SETTLE [44] algorithm. It is an optimization that essentially doubles the simulation speed. When running a simulation it is important to set the timestep for the simulation, which is how much time moves forward between each energy calculation. Additionally, combining a constraining algorithm for hydrogen bonds on top of an

exclusion water model, one in which the water radius is 0 such as TIP3P[45] or SPC[46], greatly increases computation speed due to the exclusion of these calculations in the pair list. Theoretically the lower the time that is set, the more accurate your simulation will run. However, this could essentially mean the effective real time to get meaningful insight in your simulation takes an absurd amount of time. Therefore, finding the maximum timestep possible to gain a scientifically accurate simulation is important. Due to the small movements that occur based on forces, if the time step is too big, it is possible to miss important interactions, which causes the integration to become unstable and thus violates the conservation of energy in the system and leads to simulation heating and instability. Therefore, the largest timestep a simulation can run is ~$1/10^{th}$ the oscillation time of the fastest oscillation of the system. This is known as the Nyquist limit [47]. In all atom classical molecular dynamics simulations this is the oscillation frequency of a bond between hydrogen and a heavy atom. This is on the order of ~8 to 10 fs which forces the simulation timestep to be limited to around 1 femptosecond (fs) in the absence of constraints.

Shake, removes this limitation, and allows for a 2 femptosecond timestep. It achieves this by constraining the atoms in the hydrogen bond, which essentially forces them into an optimal distance, thereby these bonds have less freedom in stretching or shrinking. These methods can really only be used if the hydrogen bond oscillation in a simulation are not important in the study being carried out.

Integration

There are several important variables used in molecular dynamics. The three most important in the dynamics are the coordinate, velocity, and the forces as these are the three variables that will determine how and where an atom will move. Out of these variables, the

forces are the only non-persistent value. At the beginning of each time step, the forces are reset to 0 and recalculated. The coordinates and velocities retain their value from the previous steps and are updated during the time step.

In general, velocities are either randomly initialized with a Boltzmann weighted distribution at step 0 of a simulation to achieve a desired temperature for the simulation, or start off at 0, representing an effective temperature of 0K. Initial positions are generated based on the configuration the user wants for their simulation. Typically this would be from a crystal structure or similar atomic resolution experimentally determined structure. Homology models can also be useful for providing the initial structure in the absence of experimental data. Although molecular dynamics follows simple kinetomatic equations for integrating the forces and velocities into new positions, the calculations are not necessarily performed in lock step through time. This is because when one calculates the new forces, it is necessary to use the positions from the previous time step, therefore the positions, velocities, and accelerations do not always line up to the same time step.

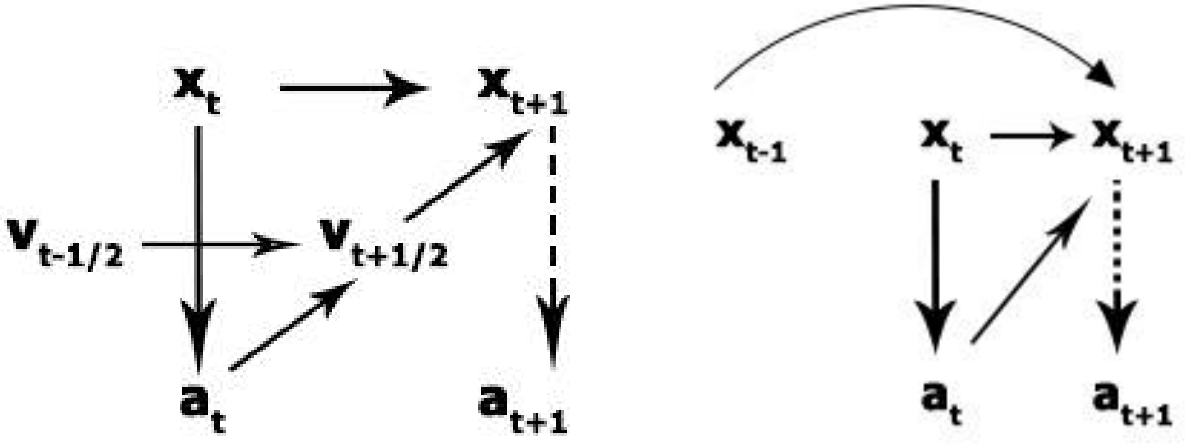


Figure 3: Left shows leapfrog algorithm which has velocity (v) offset by half a time step (t). The system evolves using a starting position and velocity. Right is the verlet method which advances through time using only the current position and the position at the previous time step.

The most commonly used approaches for integration are the verlet [48] algorithm (Figure 3 Right) and the leapfrog [49] algorithm (Figure 3 Left). AMBER uses the leapfrog protocol, but it is still important to discuss the other methods by which the system can be integrated through time. The verlet algorithm uses the current position to calculate the current forces. It then uses the current position and the previous position as well as the current forces to calculate the new position.

In the leapfrog algorithm, the positions and forces are offset by one timestep. The velocity that is used exists between the current time step and the previous timestep. The current position allows for the calculation of the new forces, which allow for the advancement of the velocity to half a timestep beyond the current time step. Using the new velocity it is possible to calculate the new position one time step ahead.

By iterating through the various different integration methods it is possible to generate long timescale simulations. This also shows that molecular dynamics is deterministic and there is no analytical solution a set of positions needs to be calculated at each time step.

Thermostat and Barostat

The discussion to this point has focused on what is required to run a basic molecular dynamics simulation in the NVE ensemble, where N, the number of atoms, V, the volume of the system, and E, the energy, are held constant throughout the simulation. By following the energy equation and using PME, the simulation maintains a constant number of atoms and the box size cannot change, and the simulation follows the conservation of energy. However, not all real world systems follow this, in fact almost no real world systems are like this but merely approximate this under very specific environmental conditions. Almost all experiments run

under conditions the pressure or temperature are controlled. There is usually a heat bath or heating up of a system, and also a container that the experiment takes place in. Therefore, simply being able to run NVE simulations is not sufficient to represent experiment conditions and therefore it is important to implement a thermostat and barostat.

When dealing with temperature, a thermostat can be used to control the heat in the system, which makes it possible to run NVT simulations, which is when the number of molecules, volume, and temperature are held constant. There is one key way to control the temperature in a system. And it centers on knowing that temperature is directly related to kinetic energy and the degrees of freedom in a system.

$$Temperature = \frac{kinetic\ \ energy}{degrees\ \ of\ \ freedom} \qquad (6)$$

$$Kinetic\ Energy = \frac{1}{2}mv^2 \qquad (7)$$

This makes it obvious that the only way to change the temperature is to affect the velocities of the atoms as it is not possible to change the parameters or number of atoms. There are a few common ways to achieve this, which results in several thermostats including: Anderson thermostat [50], Langevin thermostat [51], and Berendsen thermostat [52] which are the ones currently implemented in the Amber software. The Nose-Hoover thermostat [53] is also a commonly used thermostat in MD simulations, although not within the Amber software. It is slightly different from the other implementations as it adds an extra degree of freedom for the temperatures.

The berendsen thermostat simply rescales the velocities in a system so the kinetic energy matches the required temperature of the system. This is advantageous over other methods in that it does not reassign the velocities and is therefore reproducible run by run. The Langevin and Anderson thermostats both add a random vector to the atomistic velocities to achieve the desired

temperature. The difference is that Anderson thermostat randomly chooses an atom per time step to change the velocity to create an "imaginary collision" while langevin randomly adds a kick to all atoms. Both methods grab their random numbers off of a boltzman distribution centered around the desired temperature change needed to ensure the proper velocity, which ensures that the velocity distribution in the system closely follows the naturally occurring boltzman distribution of particle movement in nature.

In terms of barostats, the most commonly used ones are the monte carlo barostat [54] and the berendsen barostat [52]. Both of these barostats follow the same approach, which comes from basic physics, the ideal gas law:

$$PV = nRT \qquad (8)$$

Where P is the pressure, V is the volume, R is the ideal gas constant (8.813 J/mol*K), T is the temperature, and n is the number of moles of particles. When running a barostat simulation, the simulation becomes NPT, which means that number of particles, pressure, and temperature must stay constant. That means the only thing that can change is the volume. In order to do this, the box size must change. The box size differentials are calculated using the virials of the system.

There are two important parts to the virial: atomic and molecular. The calculation for pressure revolves around the center of mass of the atoms, so the virials that end up being important are those related to the non-bonded calculations, the electrostatics, and the center of mass. The virial for the electrostatics and non-bond calculation are simply the second order partial derivatives of the energy. The virials for the center of mass revolve around the relative distance of atoms towards their molecular center of mass. The pressure calculation ends up being the (kinetic energy of the center of mass – virial) / volume. Knowing the pressure and the desired pressure, it is possible to calculate the needed box size change to achieve the desired

pressure. It is important to use the center of mass of each individaul the molecule and using the same offset on each individual atom in the molecule when offsetting the system to a new box size. This maintains bond distances between atoms, which is something that must be preserved to prevent bond energies from breaking.

Reproducibility

Molecular dynamics is expected to be reproducible. The equations go in a forward fashion in that the positions calculate the forces that affect the velocities that update the positions again. This holds true for a microcanonical ensemble (NVE). However, the moment a thermostat is introduced, this point becomes moot. The most commonly used thermostats all add random number generators to the system. Thus, to have reproducibility, programs simply rely on the user inputting a random number seed, which allows the same stream of random numbers to be generated if a different user inputs the same seed [55]. However, simply being able to recreate a simulation is not always the best tactic, since it is possible that while exploring simulation space one simulation may have come to a conclusion that hardly ever occurs, or that the simulation has not fully converged. Due to this, it is not always best to be able to recreate a simulation with the same random number seed as anyone with computing power can do so. It is best to run the simulation with the same conditions and a different random number seed or simply a similar simulation started from different initial conditions in order to provide an emsemble of simulations from which average properties can be calculated. In this way one can start to approximate a typical experiment in which measurements are made on systems consisting of very large (molar) molecule counts.

Chapter 2: Computers and Molecular Dynamics

Molecular dynamics is a theory that has existed since early in the computer age. In 1966, Rahman was able to simulate 864 atoms of liquid argon for 780 time steps [56]. It wasn't until 1977, that Karplus and McCammon were able to run a 58 protein residue system[57], the first simulation that led towards molecular dynamics' prevalence in the computational chemistry drug discovery space. The reason it took 11 years to go from a simple inert gas liquid to a protein system was probably not because of the lack of theory, but more so the lack of computational power. A lot of complex theory has been developed within the field of theoretical physics. Some of these theories are simple models, whereas some are very complex, however, most of these theories do not scale well. When running simulations there are two things to consider to determine scaling: time scale and system size. It was already explained in the previous chapter that molecular dynamics scales close to O(n*ln(n)), which already shows that the more atoms that are added the more complex the computation time is and thus the more time and resources it will need. A key determinant of computational cost though, and the information that tells whether or not a simulation will be meaningful is how much time needs to be captured to get a meaningful simulation. For proteins biological activity generally occurs on a nanosecond to millisecond timescale [58] with the fastest protein folding occurring on a microsecond time scale and the slower ones occurring on a second timescale. When assuming a 2 femtosecond time step per simulation step, that becomes 500 million steps to reach one microsecond and $5*10^{14}$ time steps to reach one second.

Supercomputer basics

To begin understanding computers, it is important to understand some very simple basics. Computers operate on bits, which are simply a state of on or off (1 or 0). By stringing these bits together, it is possible to represents words, numbers, and decimals. In modern computing, numbers are stored in 32-bit or 64-bit (mostly 64 these days) values. That means for integers the values go from -2^{63} to 2^{63}. Going above or below this value creates wrap around. In general, if it is possible, an optimization for improving computational performance involves decreasing the precision in the code since using a lower precision such as 16 or 32 bits allows cramming more values in a 64-bit address, which allows for more calculations to be done on that specific register of memory at a given time. It also reduces the amount of data that needs to be communicated between CPUs and memory within a node but also between nodes.

For decimal numbers, this logic gets murkier. There are two types of representations of decimals: floating and fixed precision. By default, when a number is assigned a float (32-bit value) or a double (64-bit value) it is a floating point value. This means, that the position of the decimal relative to the whole number can change based on the value. Fixed precision on the other hand, determines how many bits are to be used for the whole number and how many bits are being used for the decimal. This puts a limit on the size on both the decimal portion of a number and the whole number portion of the number.

The last important part to understand is what a flop is. A flop is simply a floating point operation per second. Performance on supercomputers is generally measured in terms of flops to determine how many calculations can be carried out per second. Advancement of flops in a machine is not something an algorithm can change; it is something that changes as technology evolves. Generally, technology follows Moore's law, which is the idea that every 18 months the

amount of flops in leading technology doubles. Traditionally, the way to double the amount of flops was to increase the number of transistors in a chip. However, this becomes more and more complex as larger and larger chips have a higher yield fail rate [59]. Therefore, it becomes economically disadvantageous to make die sizes too big. There is thus a fixed amount of space to add more transistors. The solution for the past few decades was to simply decrease the distance between silicon molecules in chips effectively gaining more transistors inside a chip and increasing the amount of flops in a chip. As time goes on, however, it gets more difficult to cram more transistors in a chip as the atomic limit, the minimal distance between two silicon atoms, is slowly being reached.

To counteract this, parallelism has been gradually used more and more to fill in the gap needed to continue chasing Moore's law. Instead of merely relying on individual processors becoming faster, it has now become standard to add more processors into each chip. This is expanded upon even more in supercomputers, where multiple chips are strung together to reach even higher processing powers. This makes the new paradigm in computing the need for writing parallel codes to fully utilize new hardware.

The most basic universal way to optimize code is to use a more optimal algorithm. Generally this revolves around decreasing the amount of compute cycles in a code. In molecular dynamics the single biggest improvement in speed is the optimization of the non-bond code by utilizing a neighbor list that decreases an $O(n^2)$ operation to $O(n*\ln(n))$. However, beyond just using the best algorithms there are many considerations for additional optimizations, which will be discussed as the primary focus in later chapters.

CPUs

In order to begin any discussion about hardware, it is pertinent to start with the most fundamental and basic in a machine: the central processing unit (CPU). The CPU is by far the most versatile unit in a computer, since it is designed for general computation. It is designed so that it can handle any computing task thrown at it. Each of its individual threads can do complex calculations and manipulations. As time as gone on, however, chips have become more complex and more cores are being added to each processor. In today's high end chips, AMD's Epyc [60] processors offer up to 32 physical cores, and Intel's Xeon Skylake offers up to 24 physical cores with a theoretical 1030.40 Gigaflops [61]. In 1959, the CDC 6600 [62] provided a single core computing at a speed of 3 Megaflops ($3*10^6$ floating point operations per second). Today by combining many of these cores together, it is possible to achieve hundreds of petaflops (10^{15} flops) with exascale (10^{18} flops) being the next major global target.

In the past, computing was typically processor bound because the clock speed in the processor and memory were close to the same. That meant that by increasing processor speed, overall computational speed would increase. In modern day computer designs, the clock speed of the memory is much slower than the processor making most software memory bound. Calculations rely on waiting for memory to feed data into the CPU. The CPU contains a small amount of onboard memory called cache, which can typically be split into several layers L1, L2, L3. These layers go slower the higher the number making L1 the fastest. The cache is memory the CPU can instantly access and read making memory that is currently in cache the easiest and fastest to calculate. That makes it so data that is linearized and ordered in the way it needs to be accessed is faster to access than those that are randomly scattered as then the data would need to

be fetched from multiple locations and aligned as it is brought into cache, also known as vectorization.

The main advantages of a CPU is that every device in this world contains one. It is easy to write a one size fits all solution for CPUs, however, targeting for specific architectures to maximize performance is where the complexity of coding for CPU comes in. The main disadvantage is that the CPU is not tailored to solve any one problem, but is a jack of all trades. However, it is possible to use many programming techniques to overcome this disadvantage and create highly optimized codes for specific hardware.

GPUs

GPUs have been a more recent addition to scientific computing. They have existed largely as a coprocessor since the 1990s in the form of a graphics accelerator. They were largely there to offset the compute power needed by the CPU to focus on outputting graphics. The development of GPUs is largely parallel with the development of games. As gaming became more complex, GPUs also had to become more complex being able to do some simple calculations, primarily shader, which originally added shadows to textures in video games, but now are capable of complex algorithms that can strongly affect textures such as smoothing. As shaders became more complex the GPU's ability to do calculations became more complex.

A GPU's layout is different than a CPU as it is a collection of weak threads designed to be able to calculate shader work. The GPU architecture relies on all threads doing the same work, in shaders all the threads work on essentially shading pixels. Blocks of pixels generally undergo the same transformation as in a scene in a program a ray of sunlight casts on a group of pixels not individual pixels. This allows a GPU to process things quickly as groups of threads all

accomplish the exact same calculation in parallel, which is also why GPUs are so adept at the inference portion of machine learning (the feed forward algorithm) because it is essentially just matrix multiplication. The threads are grouped into warps that contain 16 or 32 threads based on the GPU architecture that work in what is known as lock-step where all the threads must be on the exact same instruction set at the same time. If one thread undergoes a different path, all the other threads must wait for that thread to undergo its path. This makes any branching (if statements) in a GPU code incredibly slow as it directly effects every thread in a warp. It is therefore faster to have a GPU thread work on dummy code that follows the same code path, than to utilize an extra thread in a warp to do new calculations.

The other important thing about a GPU over a CPU is that the GPU has faster memory reads of aggregate memory, and also because the GPU has a large amount of onboard memory, accessing memory is faster than the CPU reading from RAM. The CPU has faster individual element reads. The downside of this is that the GPU has a maximum amount of memory that can be placed per GPU, while the CPU can be infinitely expanded. Also the third fastest level memory on a GPU, the constant memory, is read-only, which makes it fast but immutable. The other downside of a GPU is that given the complexity of the project, it may be faster to do some parts on a CPU and other parts on a GPU which requires a "slow", which is relative in terms to other tasks, transfer from CPU memory to GPU memory and vice versa. Due to the many intricacies of a GPU, it was historically reserved for processing graphics, but with the invention of general purpose languages it has become possible, albeit with much more planning required, to write optimal scientific codes that leverage GPU computing power. Examples of the careful planning that is required include items such as it may typically be advised to do calculations during memory transfers to hide the latency, and rework algorithms so there are no memory

access errors that occur when two threads try to access and modify the same memory address, this adds another layer as there may need to be fences that check when all threads are done with a task before moving on to the next task.

ASICs

The last important type of processor to discuss in molecular dynamics is ASICs. These are custom chips that can be built for specific tasks. This allows for a lot of flexibility in optimizing the hardware towards what one is trying to accomplish. However, this is a very expensive solution as these chips need to be custom fabricated and designed for the task one is trying to accomplish. D.E. Shaw's group with Anton [63] and Anton 2 [64], as well as Japan's MDGRAPE machine [65-67] are the main group utilizing this solution in the molecular dynamics space. It is possible to use FPGAs which are programmable chips that can do what ASICs do, however with much less flexibility and typically significantly less performance. The biggest downside of ASICs outside of their cost is their inability to do anything other than what they were designed to do. However, what they are able to do they are usually highly optimized to solve. FPGAs are slightly better in terms of their ability to be reprogrammed, but they still require a dedicated engineering team to adapt the code.

Precision Models

One often overlooked portion of molecular dynamics is precision models. Although it may be most accurate to use as many bits as possible to solve a problem to maximize accuracy, it may not be necessary to maintain the stability and dynamics needed for the results a research problem requires. Historically, double precision floating point, 64-bit, was utilized in Amber's

PMEMD dynamics engine. However, alternative precision models were developed approximately a decade ago when porting the code to GPUs because double precision calculations were significantly slower on a GPU [68]. The idea behind different precision models is that when using usual double precision calculations, the results would always be stored as double precision and so there would be minimal accuracy loss. When running completely in single precision, the calculation would come out with 32-bits which has significantly less decimals (~7 significant figures), so if those trailing decimals that were lost were important then that piece of data would be missing.

For molecular dynamics it turns out individual force calculations are able to be computed in single precision, which is more than capable of capturing individual forces. It can capture the smallest forces as well as get enough significant figures in the largest figures. However, what is lost in using single precision, is when combining the forces. The bitwise representation of a decimal (float) in data is a 32-bit representation where the first bit is the sign then a certain amount to determine the exponent (8-bits) and 23-bits to determine the significant digits. As there is only a certain amount of precision for significant figures, this becomes a problem as molecular dynamics becomes a summation of all the forces that act upon an atom. If a small force was to be added to a big force (ex. 0.0000000000001 + 12.03415), in single precision, the smaller force is 0 in this addition and only the large force remains. This becomes an issue as there may be many small forces which may add up to a significant value, but individually adding them to a larger value makes their contributions all negligible. If forces are suddenly lost in a system, then the energy is no longer being conserved in the system and the simulation can slowly accelerate due to not interacting with all of its partners. This is the equivalent to applying an arbitrary cutoff to the simulation that is a function of the mathematical precision used. A

solution to this would be to simply sort the forces from smallest to largest so the force values are not drowned out as they are accumulated, however, such a global sort of all forces on every step would be extremely expensive and negate any gains from using single precision.

The solution that the Amber GPU code adopts is called mixed precision. This is the idea of calculating individual atoms in single precision and accumulating in fixed double precision (SPDP). The double precision accumulation allows for a wider representation of significant digits in the force values, while the single precision calculations are significantly faster as they take half the memory and floating point arithmetic to compute, they can be vectorized as 2 adds per cycle. By accumulating the single precision forces in double precision there are enough significant digits to capture both the large and small force values in the resulting forces. As an extension to this, primarily designed so that modern cost effective GeForce cars can be used is a precision model termed SPFP[68]. This is equivalent to the SPDP model described above but instead of accumulating into 64-bit double precision floating point variables, the forces are accumulated into a 64-bit fixed precision (integer) representation. This is possible because the numerical range of forces seen in an MD simulation is well defined. The reason why fixed precision is used is because the intrinsic reduction on GPU, the combination of the single forces to a single thread, needs aligned data or performance suffers heavily and additionally because the cost effective gaming GPUs lack efficient double precision support. An experimental precision model, SPXP, is also being developed which is based on Thall's paper [69], which uses 32-bit force calculations, with approximately 48-bit summation in the inner loops, and an unsigned fixed precision 64-bit summation for outer loops, resulting in close to 48 bit precision.

Vendors & Market Influences

In a very interesting set of events, memory and graphics cards are at an all-time high in cost. As of time of writing it is virtually impossible to find them at MSRP. One of the considerations when determining the hardware to buy and run simulations is the performance per dollar. Is the hardware being bought going to achieve the most optimal performance and how does that performance compare with solutions at higher and lower price point? On top of the performance per dollar, another metric that is important is the performance/watt as a less viewed side effect of running these machines is the large electricity input needed to power them.

Before discussing the concept further, it is important to highlight the top of the line CPUs and GPUs and discuss their primary functions. As of time of writing, Intel Corporation manufactures three CPU lines: consumer, Xeon, and Xeon Phi. The consumer chips are the cheapest, but the lowest core count and do not support mult-socket configurations. This is primarily due to some of the consumer chips being for laptops which require lower power consumption for battery life. Another reason is because most general consumers do not need higher core counts for every day work. The consumer chips architectures are usually ahead of the industry chips, the Xeons, which have higher core counts than their consumer counterparts. The Xeons are higher end chips with ECC memory, larger cache sizes, and support multi-socket configurations. Generally servers and workstations use these. Xeon Phi's are Intel's highest core count CPUs, however overall they have lower processing speeds. The advantages of buying the Xeon or Xeon Phi lines is the ability to run highly parallelized applications as well as still gaining, in the case of Xeon, a strong single threaded performance. Most every day workloads for consumers do not involve much parallelism, and thus do not benefit as much from high core counts. In general, gaming and media editing are by far the highest consumption tasks of a

consumer user. AMD creates two primary lines, a consumer line and the Epyc line. These are very similar to Intel in that the Epyc line is a high core count chip while the consumer line is a lower core count chip. The advantages of both lines are very similar to the Intel lines. Most other processor manufacturer's work on ARM based chips which tend to be focused on lower powered mobile and embedded devices. Most notable of these are Qualcomm Snapdragon chips which power a number of leading smart phones and tablets.

On the GPU side, there is AMD and NVIDIA. In general, gaming drives the development of GPUs as the market share for gaming is around 137.9 billion [70], while the NSF proposed budget for 2018 is only around 6.6 billion [71]. AMD has a large percentage of the GPU market and cryptocurrency mining, however, has limited penetration into the scientific compute space. This is due to AMD cards relatively cheaper cost for performance, but limited software development ecosystem. As of writing NVIDIA maintains around 70% of the market share for GPUs in total. These include graphics cards used for compute and gaming. On the gaming side is the GeForce cards that are highly effective at 32-bit calculations. Although their performance per dollar is lower, due to their increased cost, their achievable performance is significantly higher than their AMD counterparts. On the industry side is the Tesla cards that are rated for many things including reproducible results which is something that often eludes the gaming cards as precision in numerical results is not as important in the gaming space as the cards are used for output on a screen, where a wrong answer would simply cause a small artifact on the screen. While a significant concern in the early days of using GPUs for scientific compute such concerns have been mitigated by effective testing and GeForce cards now form a significant portion of the GPU hardware used in the scientific and deep learning compute space due to lower costs. In terms of parallelism, across multiple GPUs in a single node this is

governed by how many PCI slots there are. GPUs will only be able to transmit data based on if there are enough available lanes of communication which is rated by the motherboard and the processor. As data communication across a GPU is expensive, parallelizing across several GPU presents a unique challenge and between nodes an even more complex challenge. In gaming parallelism between GPUs within a node is done by having individual GPUs render different frames, however, this often causes a fair amount of artifacting. In computing, this is done by having GPUs do different tasks that do not overlap significantly, for example, individual lambda windows in a thermodynamic integration calculation.

Computer-aided Drug Discovery and the Pharmaceutics Industry

Computational chemistry, although having existed for several decades, is still relatively new in the pharmaceutical space as the costs have been high for the data it generates. In the recent past computation was mainly used to gauge smaller protein motions such as binding pocket fluctuations and structure refinement as well as static binding pocket affinity to potential drug molecules using docking [72-74]. It is only as computing has become a lot faster and cheaper that using computer aided drug discovery for a wider range of applications has been more widely adopted. With increased speeds with GPUs it was possible to start attempting complex simulations such as protein folding [75-77] and calculating absolute and relative free energy of binding. Due to the decrease cost of these computations as well as their increased accuracy with advancements in algorithmic development and refinement, the pharmaceutical industry has slowly been increasing their usage of computational methods in drug discovery.

Chapter 3: The Midpoint Method

With the advancement of both coprocessor units in the form of GPUs and Xeon Phis as well as the increasing parallelism in high end processors such as Intel's Xeon and Xeon Phi Processor lines, moderate sized simulations are now able to regularly reach a microsecond scale. The Amber Molecular Dynamics package has shown a successful implementation fully utilizing a graphical processing unit (GPU) for these simulations without relying on the CPU to supplement the computational workload. Other packages such as NAMD and GROMACS use a combination of the CPU and leveraging the GPU as a co-processor to achieve their simulation speed. When running a pure GPU simulation with Amber, the size of the simulation is limited by the size of GPU memory on board the card itself. This creates an atom count ceiling for these simulations and can run preventing large scale simulation sizes such as those of biological pathways.

Intel recently shifted its Xeon Phi line from co-processors to processors. These new processors offer a high core count and lower processing speeds with 1.40 Ghz and up to 68 physical cores. The Xeon line has also been approaching this direction with the latest Skylake processors offering up to 36 physical cores. Lastly, consumer based chips have been approaching this direction with the latest Intel i9 processors offering up to 18 cores. As computing approaches the exascale, 10^{18} floating point operations/second, there is less reliance on processing speed, and more on parallelism to approach these high compute times.

Theory

By simultaneously utilizing several techniques it is possible to make molecular dynamics which is an inherently $O(n^2)$ problem much faster. For single processor systems there are several

common techniques available to increase processing speed that are extensible to multi-processor systems. In general, the calculation of the non-bond interactions (van der Waals and electrostatics) is by far the most costly. To deal with van der Waals some sort of cut off scheme is normally used to decrease the number of pairwise interactions that need to be calculated for each atom. This is possible because forces become more negligible the farther out a pair of atoms is away from each other. The electrostatics on the other hand are computationally more expensive because they require creating a charge grid that is solved using a combination of direct pair-wise interactions and a fast fourier transform (FFT). Both of these calculations can be accelerated by using binning techniques that bin the atoms to be searched through. The bonded terms (bonds, angles, dihedrals) have typically cost negligible time since the number of interactions that are need to be evaluated are typically orders of magnitude lower than the no-bond interactions.

For the Amber's 2018 *pmemd* software release, an emphasis on memory and scalability was targeted as improvements that could be made in the *pmemd* software. Initial tests were carried out between the *pmemd.cuda* vs *pmemd* code and pmemd code scaling (Figure 4). Hotspots were identified in the data distribution timing as well as the non-bond list timing. The initial goals of the project were to have strong scaling across multiple central processing units (CPUs) primarily those of Intel Xeon Phi's Knight's Landing [78] as well as Xeon Skylake. In doing so, the internal CPU compute engine of the *pmemd* code had to be replaced.

The *pmemd* code relied on a replicated data approach where each and every processor stored and updated information on all atoms. Between each time step there would be a data distribution step that required all information on all processors to be sent and received in order to update atomic positions. This created a large overhead as more processors were used in a

simulation causing a sweet spot of processors used before scaling became much weaker. This implementation was initially advantageous due to the fact that for lower processor counts there can be perfect decomposition of the data. Each processor is able to work on only the data it is assigned and can ignore the rest of the data. Also, when the code was written, individual core processing speeds were still increasing. Over the last decade, however, processing speed has more or less remained stagnant as silicon size is reaching its atomic limits. Therefore, it has become more pertinent to rely on increasing processor counts for scaling (Figure 4).

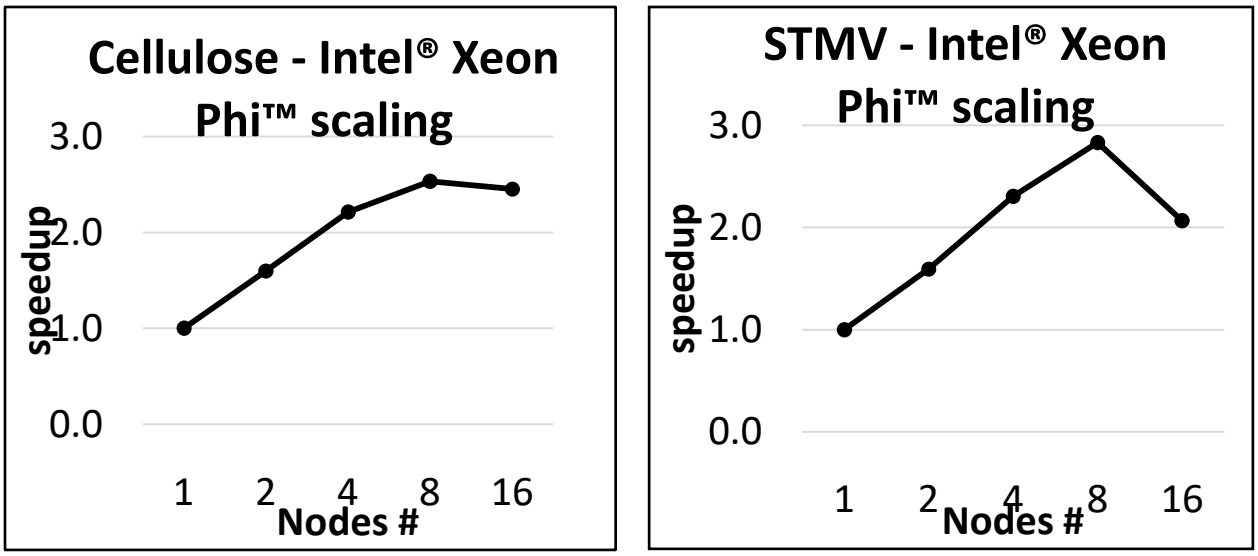


Figure 4: Scaling of Intel Xeon Phi 7250 processors for Satellite Tobacco Mosaic Virus (STMV) (1 million atoms) and Cellulose (400,000 atoms) across multiple nodes.

To rectify these shortcomings, a 3-D domain decomposition method was developed for *pmemd* with a focus of maximizing stability on Xeon and Xeon Phi CPUs. The most common domain decomposition is the 8^{th} shell method [79] used in GROMACS and the midpoint method [80], a method originally devised by the D.E. Shaw research group for their molecular dynamics engine, Desmond. The midpoint method, the method selected for this work, is a neutral territory

3-D Domain Space decomposition, where the processor chooses the non-bond interaction to calculate based on where the midpoint of the two atoms lies. The method has been used in GENESIS [81-83] and an IBM project called Blue Waters [84]. Goals of the project included keeping the high performance single core performance of *pmemd* while simultaneously improving scaling performance. Additionally, there was a need to decrease memory costs by removing the need to replicate data across all processors as increasing atom count into the millions uses up gigabytes or more of memory and introduces serious I/O bottlenecks.

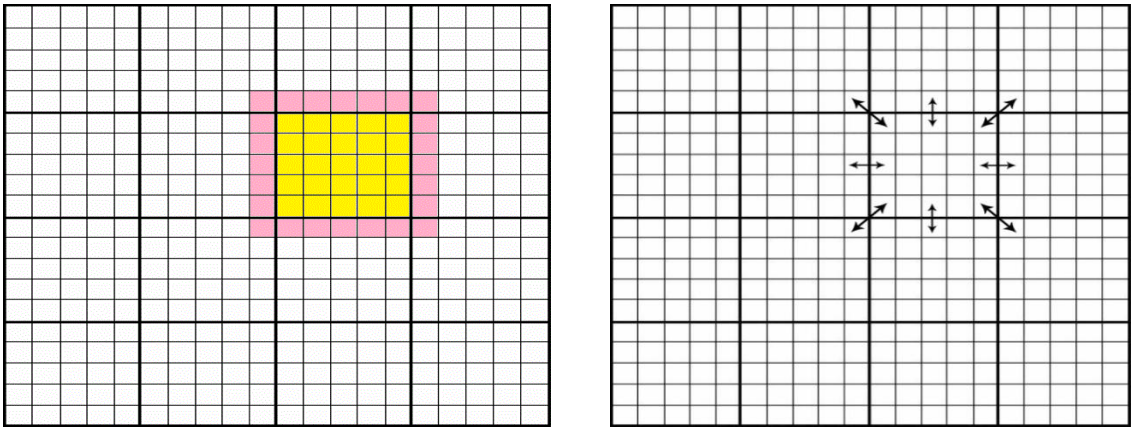


Figure 5: Left shows the domain decomposition where yellow is atoms owned and pink is the ghost regions that come from different processors. Right shows the communications across adjacent processors.

The primary code design revolved around giving each set of processors an integral cubic number of bins whose size were slightly larger than half the cutoff (Figure 5). By allowing the system to have a one bin buffer outside of the space they owned it allowed the needed non-bond calculations across different processors to be calculated. Communication was only needed on the atoms in the buffer bins, which were only required to be sent to the adjacent overlapping processor blocks. This change decreased the amount of communication needed to be done in the system from O(n) to $O((N/P)^{2/3})$. Additionally for bonded terms only the lead atom needed to be tracked. For bonded terms the lead atom was the first atom, for angle terms it was the middle

atom, and for dihedral terms it was the second atom. When the lead atom moves processor space then the bonded parameter set would also need to move with the lead atom.

Without any baseline optimizations, the communication costs were shown to heavily decrease as processors went up when using a non-bond only system when compared to the original *pmemd* implementation (Figure 6).

Initial Work

Previous work on the replicated data approach for *pmemd* on the KNL architecture showed that taking advantage of AVX-512 optimizations resulted in a 2x increase over baseline Amber16 performance. However, performance on smaller systems such as Cellulose showed a mere 2% increase in speed. Previous work used strategies such as neighbor list vectorizations by adding linearized list for atoms in buckets, increasing loop length of nondependent loops, and adding compress store optimizations (shown below) available with AVX-512.

Table 1: Left shows an example of compressed store, right shows an example of conflict store. The conflict in the array indicies is the dependency of the evaluation of the conditional statement before calculation, which makes it difficult to traditionallly vectorize. AVX-512 can inherently vectorize these scenarios.

Compressed Store:	Conflict Update

J=1
Do i=1, N
If(A(i) .gt. x) then
B(j)=A(i)
J=j+1
End if
End do
	do jn = 1, dihed_cnt
	i = dihed(jn)%atm_i
	……..
	if (gbl_atm_owner_map(i) .eq. mytaskid) then
	frc(1, i) = frc(1, i) + fxl //conflict in i
	frc(2, i) = frc(2, i) + fyl
	frc(3, i) = frc(3, i) + fzl
	end if
	end do

Another optimization, conflict update and compressed store (shown above), is able to be vectorized using AVX-512, and is primarily used in the bonded portion of the code. The last major change that came with the previous work was improved load balancing. In the original *pmemd* code, the atom arrays were copied across all MPI tasks. Doing this allows for perfect decomposition of splitting the work into number of atoms / number of processors. However, it is not efficient as each processor needs to do more work than it needs because arrays initialize at the length of the array, and the arrays need to be reduced across multiple arrays of the same size. These methods do not scale. In order to fix this a min/max range that each MPI task used was found and that range was used for initialization and reduction. These results gave significant increased speed in larger systems.

These changes to *pmemd*, although significant for larger workloads, had issues at higher core counts due to communication problems. With the *pmemd* midpoint code, along with changes in the internal compute engine, the same optimizations were made, particularly those of vectorization, OpenMP threading, and mixed precision modeling. Primary focus in optimizations were taken in the non-bonded portion of the code, which takes a majority of the compute time, which consists of the neighbor list building, van der waals forces, and electrostatic forces.

Upon examination of this implementation by splitting the system into different MPI tasks with relatively the same amount of atoms it is possible to check the relative parallel efficiency of the system:

Table 2: Lists the major functionalities in a basic molecular dynamics simulation and their efficiencies assuming all processors have roughly the same number of atoms. In this case, they should all have relatively the same amount of time in all tasks in perfect decomposition. If the task requires calculating information outside of the decomposed atom count, then scaling would not be efficient, and therefore would be a hotspot for optimization.

Function Name	2000 steps 4 MPI 24K atoms 6000/MPI	2000 steps 68 MPI 408K atoms 6000/MPI	Parallel Efficiency (100% is ideal)
Data Distribution (MPI)	**7.96**	**113.14**	**7.04**
Shake	1.41	2.11	66.82
RunMD (management)	4.9	9.76	50.20
Other (overall)	0.78	1.94	40.21
Create Coordinate Index Table	**0.64**	**23.2**	**2.76**
Neighbor List Build	9.7	13.55	71.59
NonBonded Calculation	39.94	38.91	102.65
Exclude mask	2.18	4.1	53.17
Other routine of direct force	**3.84**	**11.38**	**33.74**
Packing for direct force	1.04	3.84	27.08
Neighbor List Packing	**0.43**	**5.34**	**8.05**
1D bspline (Recip.)	2.84	4.3	66.05
Grid charges (Recip.)	4.81	5.72	84.09
Scalar sum (Recip.)	0.32	0.32	100.00
Gradient sum (Recip.)	5.9	7.01	84.17
FFT (Recip.)	**11.29**	**19.54**	**57.78**
Total	97.98	264.16	37.09
ns/day performance	3.55	1.29	

It is through this analysis, that it can be noted that the major issue with the original implementation of the *pmemd* code is the data distribution task. The system still needs to send information of all atoms to all other processors during every step, which essentially takes up a large portion of compute time.

A proof of concept using the *pmemd* engine was made to test a change in the core algorithm from a replicated data approach to a 3-D domain decomposition, midpoint method approach. By using an inert liquid, it is possible to only make changes to the van der Waals and kinetomatic portions of the code. By analyzing the effect of data distribution, it is possible to

check the impact of the processor decomposition. When changing to a more efficient algorithm, the overall communication time decreased significantly due to the immensely decreased amount of data that needs to be transferred. (Figure 6)

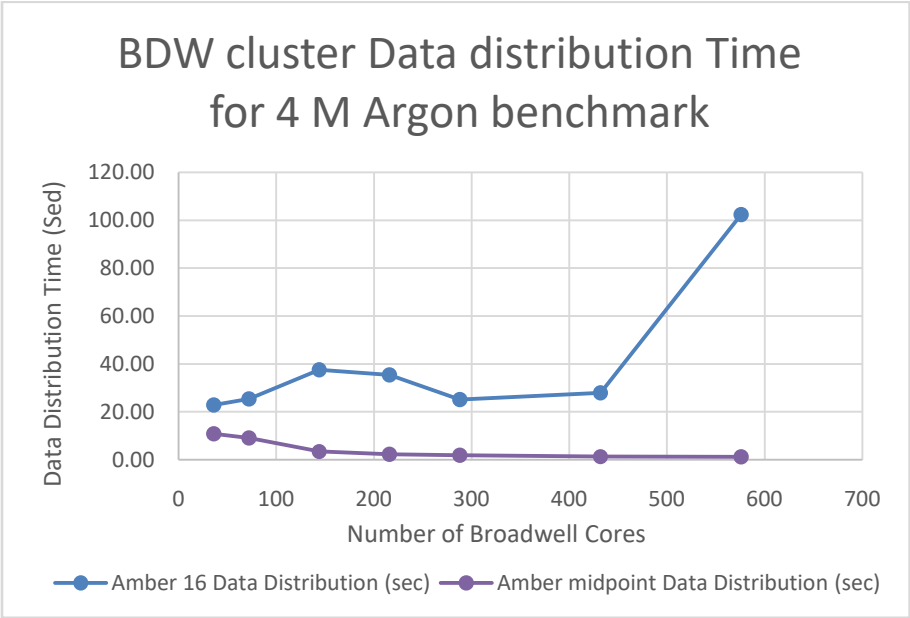


Figure 6: An initial run using 4 million atoms of argon to test the efficiency of the data communication across hundreds of CPU cores. Timing is the total time spent in communication during the simulation.

Communication

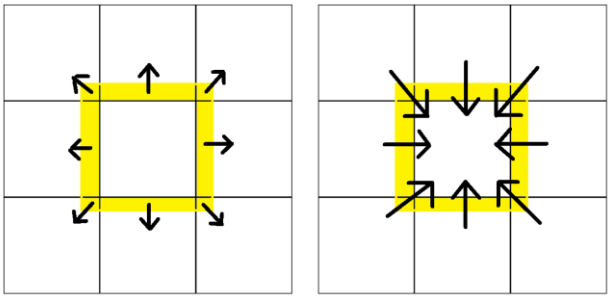


Figure 7: Figures show a processor and its neighbors. Yellow represents the ghost region of the central processor. Left shows an inside out method where ghost information is sent to the respective neighbors. Right shows an outside-in method which is used for the population of the ghost region.

Minimizing the number of message passing interface (MPI) communication calls to only when necessary reduces communication overhead. Reducing an all-to-all communication for the

majority of the code to just the nearest 26 neighbors helps tremendously decrease overall communication overhead. In AMBER only a few types of communications are needed: an inside-out send to send data from a local region to ghost regions, an outside-in send to send data from the ghost region to a local region, a per-atom send to send atom information to a different processor, and an all-to-all send to send information needed to all processors.

For the midpoint method to work the inside-out and outside-in methods are needed. Both of these processes add a copy of an atoms information onto another processor and the data originally maintained by the local processor remains, but may be altered. This allows for atoms to interact with atoms that are owned by a different processor. In the pmemd implementation, the inside-out communication (Fig 7 right) refers to when an atom's information is transferred from its local processor to the ghost space of another processor. The shared regions are copied from the processor that owns the information to the ghost region of a bordering processor. This is commonly used to transfer parameter data, including mass and coordinate of an atom to neighboring processors to be used in the non-bond calculation of the code. The opposite, an outside-in approach (Figure 7 left), is needed for the transfer of force information back to the local processor. This information is needed for the local processor to keep track of where an atom is going to move in a simulation. In molecular dynamics, the kinetomatic portion of the code which involves the movement of the atoms only needs information about the current position, velocity, and the force. The local region already has the position and velocity of the atoms, so only the partial forces that have been calculated upon the ghost region need to be transferred back to the atoms local space.

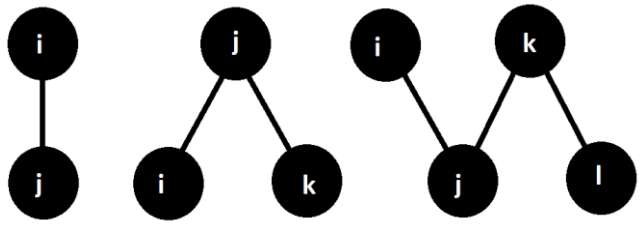


Figure 8: In bonded terms, the atoms are labeled I, j, k, and l. In bonds, atom i is the lead atom. In angles, atom j is the lead atom, and in dihedrals, atom j is the lead atom. The lead atom's location determines the processor in charge of calculating the bonded term.

An atomistic send is also needed for cases where individual atoms and/or parameters need to be moved to a specific processor. This send method allows the packing of an individual atoms entire parameter and data set, including but not limited to velocity, coordinate, force, mass, and parameters. This is useful in two areas of the current implementation: barostat and bonded code. The barostat requires the change of the box size, which causes a shift in the atoms and their potential position in their local space. This can potentially cause issues with breaking the local space of the processors, which may require a few atoms per processor to be shifted to their correct processor. The bonded code uses direct send for a different reason. In order to not double calculate the bonded code (bonds, angles, dihedrals), the code determines a lead atom in these bonded regions, for bonds its atom 1, for angles and bonds it's the second atom (Figure 8). When the lead atom moves to a different local processor, the parameters for the atom set moves too.

The most expensive portion of the MPI communication is the communication across the FFT. This is expensive not because of the data that needs to be sent, but the inability for the scaling of the communication. The Particle Mesh Ewald (PME) consists of solving a grid, which requires several all-to-all MPI communications. The data involved in the FFT transfers is much smaller than the ones used to transfer atom information, but due to the number of MPI processes's required, the more MPI tasks are in a simulation the more expensive this portion of

the code becomes. Therefore, a future optimization involves setting aside 1/7 of the processors to purely do the reciprocal charge calculation of the code and nothing else.

Vectorization

In order to reduce the memory footprint, the decision was made to carry one copy of the coordinate, force, and velocity arrays. In order to remain cache efficient there was a need to sort the atoms at each neighbor list build. This helped with the vectorization of the electrostatics code when building the initial charge grids for particle mesh ewald (PME). A data type containing bucket and atom coordinates allows for efficient traversal when building the neighbor list.

The optimizations provided by AVX-512, which allows for 512-bit SIMD instructions, helps further vectorize the code by allowing efficient compress store and conflict update code patterns. The conflict update code pattern occurs heavily in the bonded codes as well as in the electrostatics code, while the compressed store occurs primarily in the electrostatics code. By having compile time vectorization of these routines, there is a small speed gain. This is particularly helpful in the electrostatics code as the routines that set up the charge grid before the fast fourier transform take up a nontrivial amount of computation time.

OpenMP

The use of multithreading is very advantageous in Intel systems as they are already configured to allow for hyperthreading. By adding the use of OpenMP in the midpoint codes processing of data in the non-bond calculations there was a 30% performance improvement.

Precision Modeling

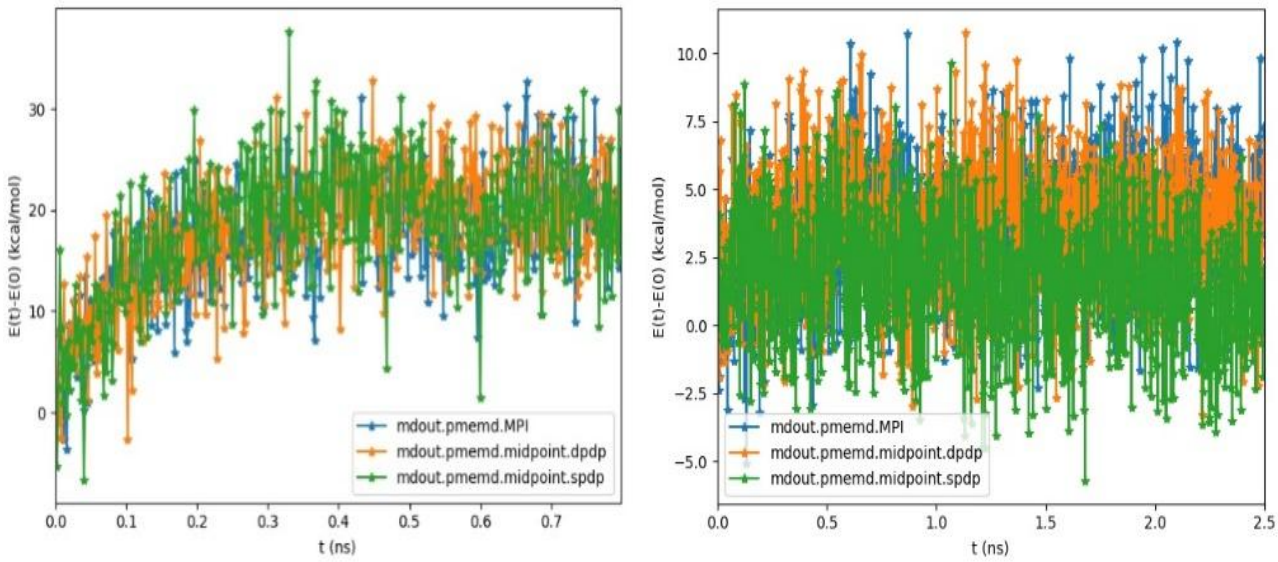


Figure 9: Stability of single precision vs double precision simulations in Cellulose (left) and Factor IX (right). Midpoint single precision is in green, midpoint double precision is in orange, and non midpoint code is in blue. In general energy is conserved between the codes over the course of a couple nanoseconds with no overall increase in heat.

In the midpoint implementation, an option was implemented that allowed single precision force accumulation with a double precision summation for all portions of the non-bonded code (SPDP). This includes the charge grid and all FFT communications being carried out in single precision. A double precision accumulation is used to prevent large integer numbers from drowning out the very small decimal values in accumulation. An alternative to achieving this in purely single precision would be to sort from smallest to largest and then accumulate from the smallest value to the largest value. Single precision accumulation was not incorporated in all aspects of the code due to the already minimal time spent in the bonded routines. The addition of mixed precision gives the code a 20-30% boost in most workloads as further vectorization is achieved with the ability to essentially double the amount of calculations that are done to calculate the non-bonded forces. However, these changes add slight precision differences in the energy values on top of the pre-existing precision errors that occur based on the order of addition when many MPI tasks are involved.

Table 3: Single point error calculations were taken out on step 1 between DHFR, Factor IX, Cellulose, and STMV and the values were compared against the CPU non-midpoint version of the code. The magnitude of the error deviations fell within the error range produced on the GPU SPFP and DPFP error range, which has been proven to conserve energy over long timescales.

Max Deviation of Errors: (GPU results from JCTC 2013)

	DHFR	Factor IX	Cellulose	STMV
Midpoint SPDP	8.6 E-04	1.9 E-03	3.8 E-03	4.0 E-03
GPU SPFP	3.4 E-04	2.2 E-03	1.9 E-03	4.0 E-03
Midpoint DPDP	5.0 E-08	4.7 E-07	4.9 E-07	5.0 E-07
GPU DPFP	6.0 E-08	2.1 E-06	4.6 E-08	9.4 E-08

RMS Deviation of Errors: (GPU results from JCTC 2013)

	DHFR	FactorIX	Cellulose	STMV
Midpoint SPDP	5.4 E-05	1.1 E-04	1.4 E-04	1.9 E-04
GPU SPFP	2.3 E-05	1.2 E-04	7.8 E-05	1.3 E-04
Midpoint DPDP	1.9 E-08	2.0 E-08	2.0 E-08	2.0 E-08
GPU DPFP	1.5 E-09	1.5 E-09	1.3 E-09	2.1 E-09

Performance

Simulations were validated with both long and short timescale simulations on Cellulose, JAC protein, poliovirus, and satellite tobacco mosaic virus. These were chosen to fill a role of various simulation sizes. Long timescale simulations (Figure 9) were used to show energy conservation in NVE simulations. Short time scale simulations were to assess accuracy of results compared with Amber16 and Amber14 results. Additionally, single force calculations

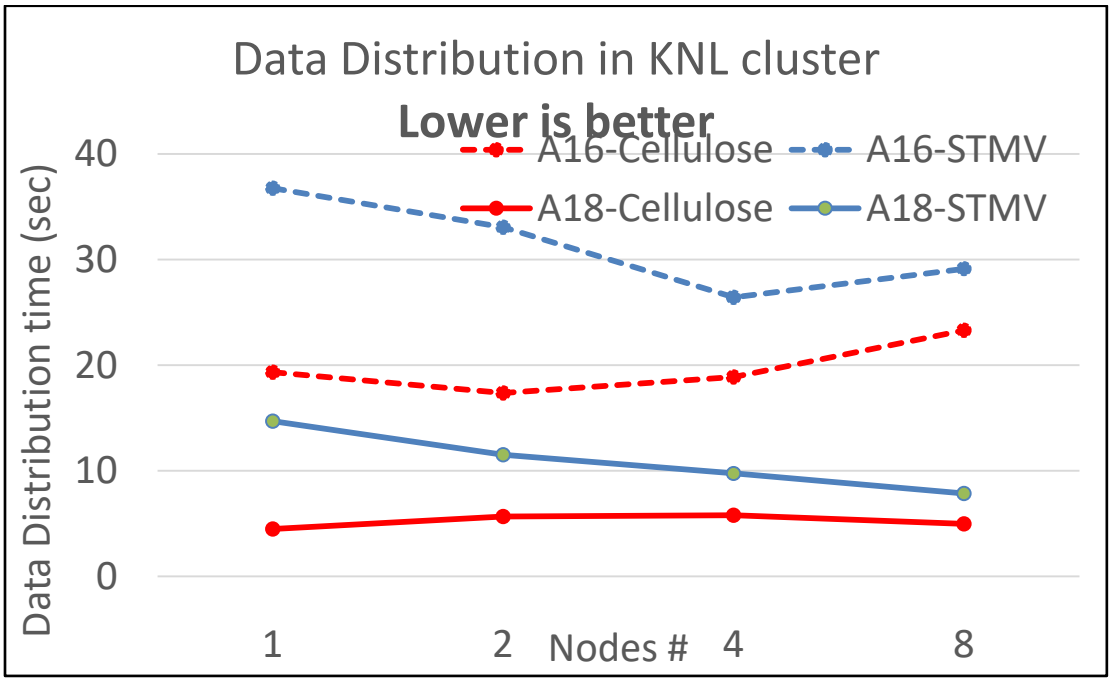
were run to check RMSD between simulations and these stayed within a reasonable value (Table 3).

Results between *pmemd* and *pmemd* midpoint differed slightly in the electrostatics if the PME grid is split in an odd number due to the differences in charge grid splitting. However, the electrostatics energies match if both use even grid splitting. In order to accommodate increased precision for older hardware, Amber14 transformed the spatial arrangement of atoms by shifting all axes by half a box length. The midpoint code does not contain this shift. This creates slight numerical differences when the charge grid contains an odd number of divisions causing one grid cell to be split in half. However, these results were validated using the long timescale simulations.

In addition, considering the core focus of the project was towards the Xeon Phi processor line, it can be seen that *pmemd* is comparable to *pmemd* midpoint on Xeon processors for lower node counts. This is likely due to the fact that the code is optimized for Xeon Phi's larger L2 cache and lack of an L3 cache. This allows for better vectorization in a localized area, however, it weakens the ability for both shared cache and random memory access. Considering Xeon's L2 cache is smaller, it likely takes a performance hit on these optimizations. However, Xeon gains performance on larger nodes likely due to *pmemd* midpoint's better scaling over many cores.

Data distribution of the midpoint code performed 2-4x better than the Amber 16 *pmemd* with more significant performance gains at higher core counts (Figure 10). This is expected due to the inefficient replicated data distribution method in the older code. Overall performance across one to many nodes also doubled the baseline code (Figure 11). This is expected as even though the majority of the code performs better than Amber 16, the electrostatics code which

takes approximately 1/7 of the time still performs all-to-all communication and therefore does not scale at higher core counts.



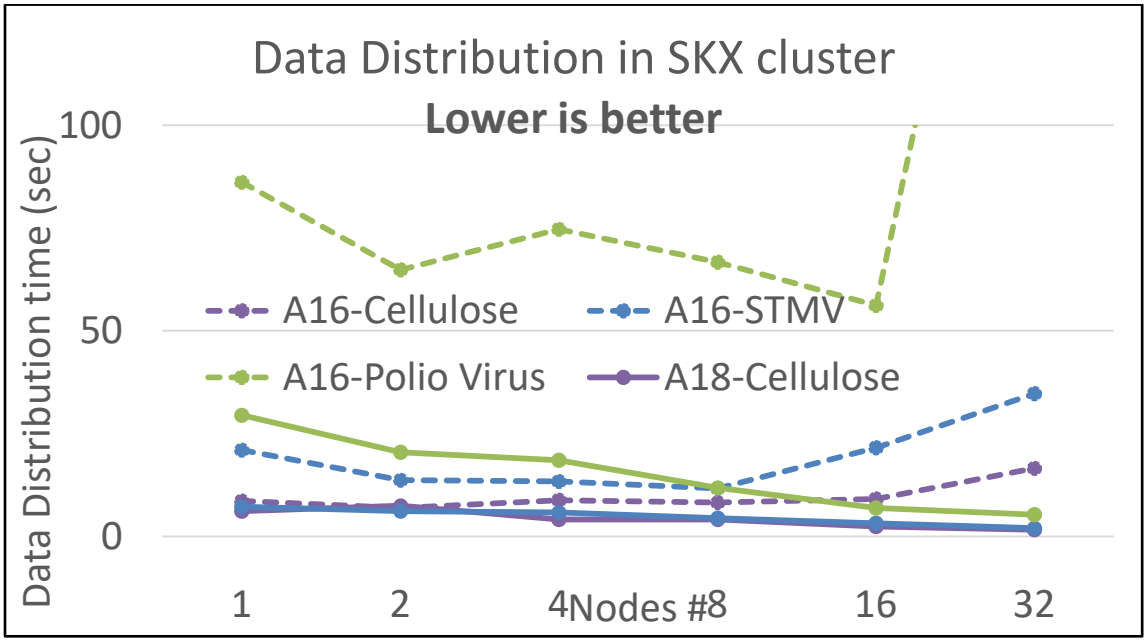


Figure 10: Data distribution times were compared on Skylake and KNL nodes after a cumulative 800 time steps, showing overall data communication to be significantly better than the Amber 16 baseline scaling even through several hundred cores.

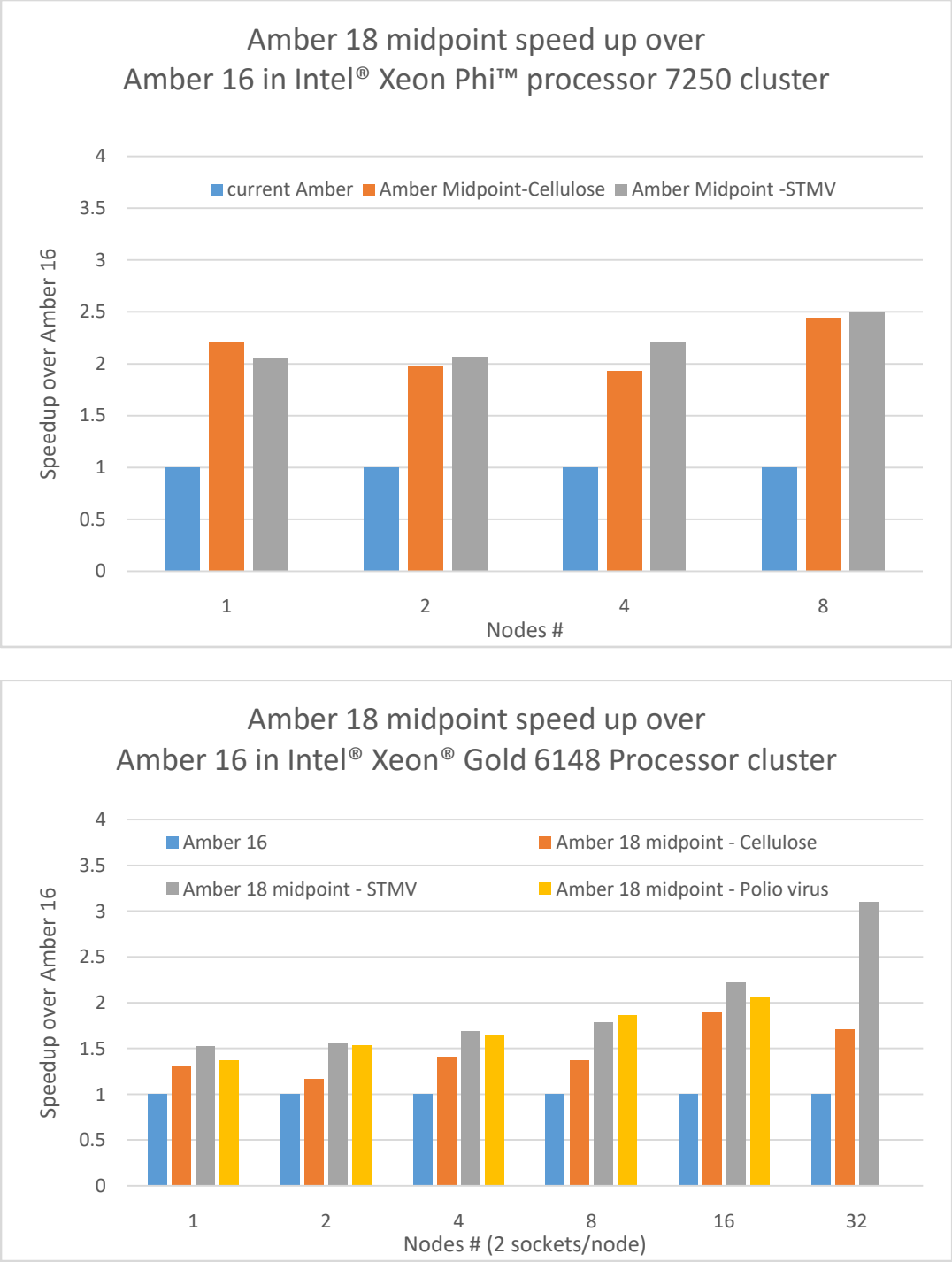


Figure 11: Performance data was collected over a baseline Amber 16. These numbers were collected over 800 iterations at a 2 fs time step using shake and no print outs. Overall, the midpoint code achieves significantly better scaling than the Amber 16 baseline especially at larger system sizes and larger core counts. Amber 16's data distribution time greatly increases at very large core counts which makes the 3-D domain decomposition method much more efficient and optimal.

Features

As of the release in April 2018 for Amber 18, the *pmemd* midpoint code supports NVE simulations, shake, and the langevin thermostat. The release version of the code has memory scaling issues due to a lack of dynamic memory allocation for the ghost region. Thus, a large amount of memory is allocated for all arrays that span the number of processor atoms and ghost atoms. For the first update slated for the midpoint code, the berendsen barostat, Monte Carlo barostat, langevin thermostat, and berendsen temperature coupling scheme will be added as well as dynamic memory allocation. The berendsen barostat and langevin thermostat used in conjunction decrease speed by around 50% due to the cost of using the Gaussian random number generator as well as the additional calls to the neighbor list that are needed for reorganizing the atoms for the barostat. As of early testing, due to the 3-D decomposition of the system, when a neighbor list is invoked, the contents for all processors change. This causes the need for saving states when using a Monte Carlo barostat which is the main computational deficiency in the Monte Carlo barostat implementation. In general, however, the NPT scheme using a Monte Carlo barostat and berendsen temperature coupling runs faster than the non-midpoint based code. Simulations run with the berendsen barostat and langevin thermostat match the non-midpoint version with a reasonable force error from round off.

The performance versions of the thermostat, the weak coupling thermostat, as well as the performance version of the Monte Carlo barostat, are slated for a late 2018 patch. Additional features including but not limited to restraints, constraints, thermodynamic integration, and replica exchange simulations are slated for early 2019. With a complete commonly used feature set it is hoped that the midpoint version of *pmemd* will become the de facto version of Amber as it scales significantly better than its predecessor.

Conclusion

With a combination of hardware improvements on top of optimizations to the AMBER code to better fit evolving hardware, it has been possible to greatly increase performance (Figure 12). As software performance is not only about hardware, but about the algorithms underlying the hardware as well as the optimizations tuned towards evolving hardware, it becomes a never-ending quest to reach the optimal performance of each hardware generation. As AMBER's age began to show with the passing of the fast-single core CPUs into a multi-threaded, mult-core world, changing the underlying computational engine's capabilities to be better suited for multi-core was required to significantly unlock the performance that these high-end chips are capable of.

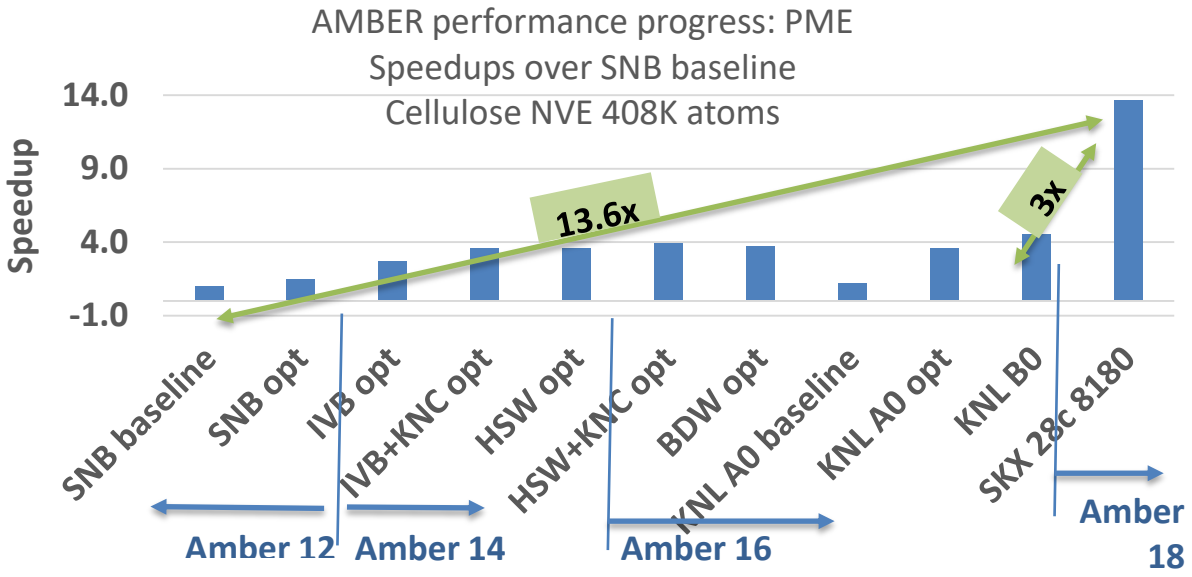


Figure 12: Through a usage of hardware updates as well as software updates, the Amber software has had over 13x increase in speed since 2012. Between Amber 12 and Amber 16 there was only a 4x increase in performance speed which was largely gained through hardware updates. KNL which has 68 physical cores is only slightly faster than the Haswell (HSW) Xeons which only had a maximum of 18 cores. With the midpoint update a 28 core Skylake is able to outperform Amber 16's 68 core KNL performance by 3x.

Chapter 3, in part is currently being prepared for submission for publication of the material. Lin, Charles; Malas, Tareq; Bhuiyan, Ashraf; Yeh, Allan; Walker, Ross C. The dissertation author was the primary investigator and author of this material.

Chapter 4: Thermodynamic Integration

One of the holy grails of computational drug discovery is the ability to calculate free energies of binding in vivo. In traditional drug discovery, high throughput screening can be carried out on drug targets to determine whether or not a target is a good compound or not, but this only works if the molecules already exist. By using computers, it is possible to skip the step of synthesizing new molecules that may be expensive and take a significant amount of time to synthesize, and instead just use computing time and power to test out the free binding of new molecules before testing them in a test tube.

Many different methods of estimating free energies exit ranging from simplistic, but fast calculations such as MM-PBSA[85], to intensive and rigorous calculations such as Thermodynamic Integration. The tradeoff between accuracy and time becomes important as these calculations can takes months of compute time on single-core CPUs. However, with the popularity of GPU computing, these calculations have been making their way to GPUs. Schrodinger's FEP+[86-91] has been the industry de facto solution for calculating free energies as it is both fast and relatively accurate but mainly due to it until recently being the only option on GPU. With Amber 18's release, the ability to run Thermodynamic Integration on GPUs has been released[92, 93]. Taking advantage of GPUs as well as the inherently parallel nature of these calculations which involve many different calculations to reach convergence, it is possible to greatly decrease the real world wall time of the calculations by running the many different windows and stages on different GPUs.

Thermodynamic Integration

Free energy is generally calculated with the energy of a ligand and the energy of a ligand and protein complex in solvent. This can be done because the energy of transformation of ligands in solvent and complex in solvent is the same as the difference in free energy between the free energy of binding of the two ligands (Figure 13).

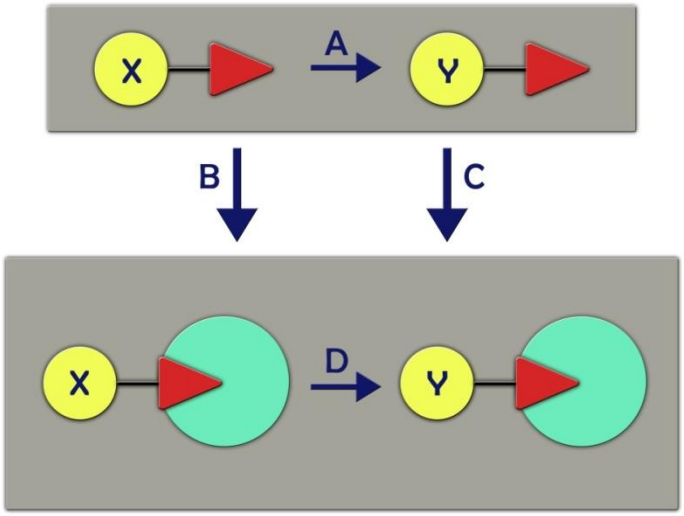


Figure 13: It is generally easier to do relative binding free energies because the absolute binding free energies (B and C) are expensive to calculate. It is possible to calculate the relative change C-B by calculating D-A which require a ligand transformation and a complex transformation that are both easy to perform.

For this to work, the majority of the atoms in the ligand must stay the same, and the differing atoms are placed in a softcore region where the charges of these atoms are selectively turned on or off. Due to the need to get phase space overlap between the two systems in these calculations, a significant amount of sampling is needed for the systems to converge. By setting weights for the softcore regions[94], regions where atoms are not shared, using lambda, it is possible to set weight to the original state stronger (using a smaller lambda) or the new state stronger (using a larger lambda). This allows for further phase space exploration and overlap in the calculation.

The actual calculation of the relative free energies is done during post processing. In Thermodynamic Integration, dV/dLambda is the derivative of the energy relative to lambda. Because V is relative to the lambda value, where V_0 is the starting state, and V_1 is the end state, having more windows therefore makes the approximation of the curve more accurate.

$$V = (1-\lambda)^k * V_0 + (1-(1-\lambda)^k * V_1 \qquad (9)$$

By calculating the integral under dV/dL it is possible to calculate out a free energy change for the state. By summing together the states that make up the ligand change and complex change it is possible to subtract the two to get a relative free energy change of the ligand. Multistate Bennett Acceptance Ratio (MBAR)[95, 96] uses the simulations run in TI, but calculates what the energy would be at the given lambda. The advantage of MBAR is the decrease in variance as well as being able to calculate convergence in the free energy system.

TNNI3K System

Cardiac troponin I-interacting kinase (TNN13K) is believed to play a role in cardiac disease as it is only expressed in the heart. In mouse models,overexpression has shown to increase hypertrophy and heart failure[97], meanwhile knocking out TNN13K reduces progression in cardiac injury. As the full mechanism of TNN13K in its role in heart disease is unknown this suggests the inhibition of TNN13K to be a likely target to help ease the effects of heart ischema and dilated cardiomyopathy[98]. To test the binding efficiency of potential inhibitors in silico the co-crystallized structure of TNN13K (PDB ID: 4YFI) with an inhibitor, 3-((9H-purin-6-yl)amino)-N-methyl-benzenesulfonamide, was used as the basic structure for the simulations[99] (Figure 14). The hydrophobic pocket contains pi-pi interactions between the

purine and tyrosine 541, and threonine 539 and lysine 490 create strong electrostatic interactions with a sulfamide structure.

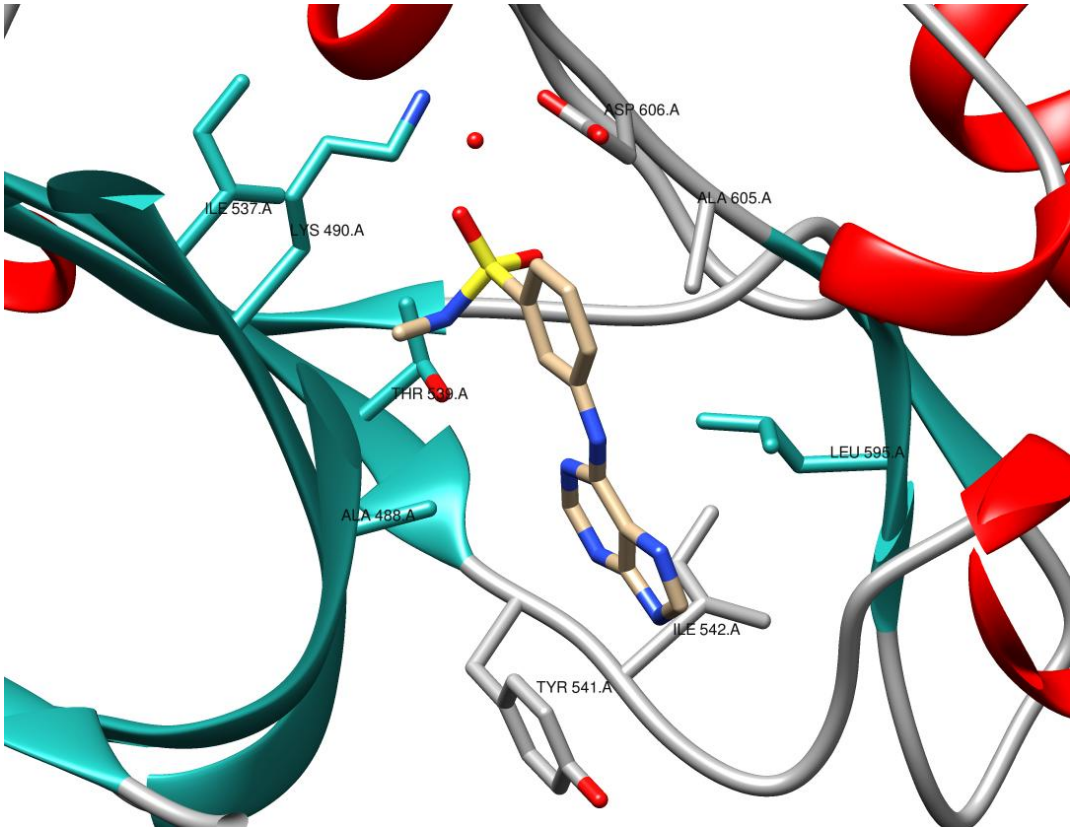


Figure 14: TNNI3K with inhibitor 3-((9H-purin-6-yl)amino)-N-methyl-benzenesulfonamide

Setup

Simulation setup was generated from the TNNI3K dataset from GSK. Transformations were created with Amber using the GAFF forcefield and TIP3P water. During preparation, steps were taken to ensure matching atom types between the molecule and the forcefield. When bonding information did not work for antechamber, bonding was manually modified in the mol2 file. As *leap* does not read bond information, the user has to specify the bond type. Ligands were charged using AM1-BCC partial charges using *antechamber*. The system used GAFF and FF16SB Amber forcefields for parameters. Once the ligands and the protein were parameterized,

one ligand for each the before and after transformation were added in the same files for ligand simulations, and the two ligands with an included complex were added for the complex transformations. 100 steps of minimization, followed by 10,000 steps of heating and pressure were run on the system. Simulations were run in 11 windows from 0.0 to 1.0 in 0.1 steps. Simulations either contained the 2 ligand transformations or a complex containing the two ligand transformations going through three transformations from uncharged, bonded, and charged. This resulted in 66 simulations per transformation. All minimization was done in the MPI version of *pmemd*.

Ligands were heated on GPU, followed by 20,000 steps for production MPI CPU for stability. This is because in the production step a barostat is included so there was a chance that the box size may decrease significantly due to the removal of water. This could potentially cause issues on the GPU as large changes in box size will cause memory errors in the GPU code. If the starting structure was inadequate for certain lambda windows as lambda windows affect the resulting forces, an additional 100 cycle minimization was carried out at that lambda window.

Complexes were run exclusively on GPU after the minimization step, and also run for 5 nanoseconds each. Simulations were mostly run on *pmemd.cuda_SPFP* and if there was a memory issue then *pmemd.cuda_DPFP*, and if there was a segfault issue, then CPU. The DPFP build had a chance to fix issues the SPFP build experienced if the overflow was due to a single precision floating point. It was likely that simulations run in DPFP could also have overflown due to the accumulation being a fixed precision accumulation. This resulted in the need to run in *pmemd.MPI* to overcome issues with highly strained systems. Generally only the endpoints (0.0 and 1.0 window) were needed to be run in the MPI build. In this system, only one of the 2310 simulations needed to be run in the MPI build.

After the initial equilibrating, the ligand simulations were run in production for 2.5 million steps at 2 femtoseconds resulting in a 5 nanosecond simulation. This was run at 300K using a Monte Carlo barostat kept at 1 bar. Dvdl values were printed every 1000 steps resulting in 2500 values, of which the first 1250 were discarded to assume stability in the system. This left 1250 values to make the reference ddg values which were calculated using exponential curve fitting to determine the integral for each window. These were then summed to generate the ddG values.

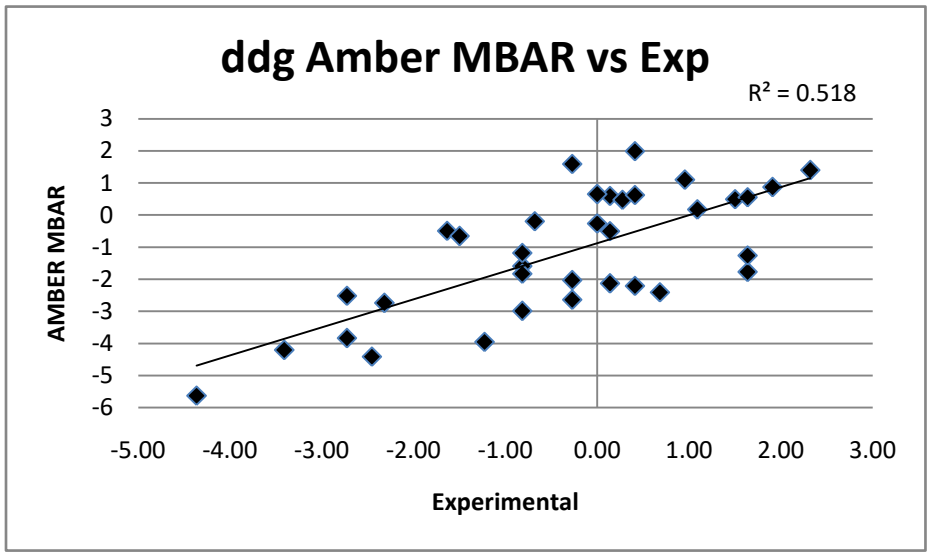
MBAR values were printed every 10,000 steps due to the decrease in performance from MBAR with the first 125 values thrown out. This left 125 values to calculate MBAR. When calculating MBAR, alchemical analysis software was used. The majority of these values gave SciPy warnings that xtol is too small, but results showed that despite warnings the simulation converged.

Discussion

ddG values were calculated using transformations obtained in a previous GSK data set[99]. The overall ddG energies that were reported for each transformation were compared between experimental, AMBER TI, AMBER MBAR, and FEP+.

Data was analyzed for ddG against experimental values with AmberTI, MBAR, and FEP+. For FEP+, raw values and values with cycle closure correction were analyzed (Figure 14). In general, for transformations, it is expected that when a ligand transforms from A -> B -> C -> A, the overall energy change is 0. FEP+ has a cycle closure correction [100] method that minimizes the error in the cycle transformations. When analyzing individual ddg values, results seemed to indicate that all three performed fairly well, with AMBER TI and AMBER MBAR

performing slightly better. The methods achieved relatively the same results between each other with AmberTI and FEP+ having a 0.702 correlation and AMBER TI and AMBER MBAR having a 0.881 correlation (Figure 15). However, these values are not indicative of performance as what is important to be measured is the actual free energy states. However, with relative free energies it is not easy to calculate absolute free energies. Instead it is possible to create a baseline free energy and calculate relative free energies against that baseline.



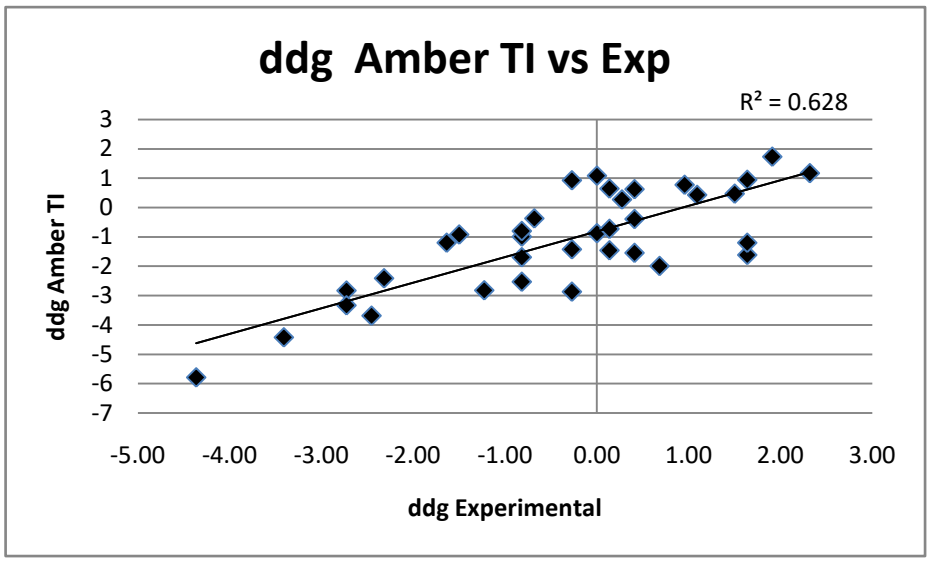
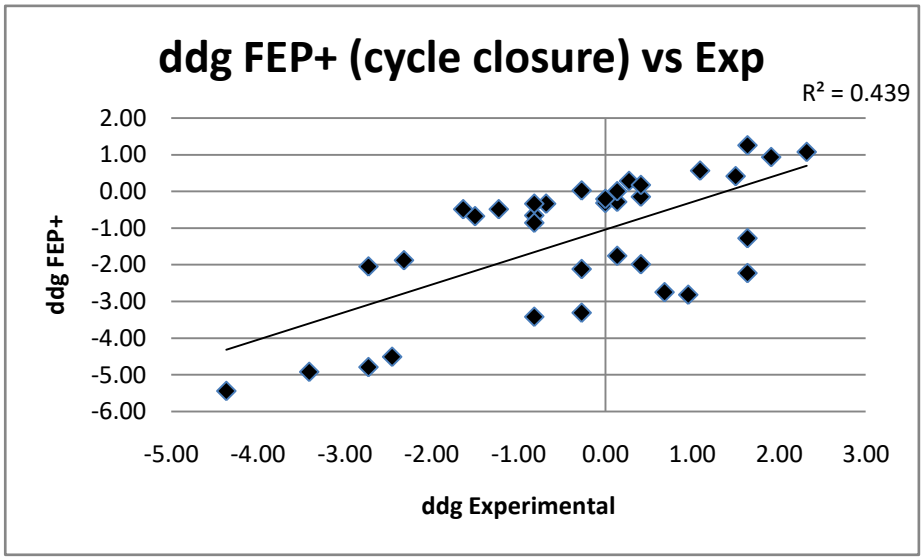


Figure 15: These two graphs show individual edge map transformations for the TNNI3K system. Both results use the same trajectories, but MBAR uses the MBAR post processing to get its ddG values while TI uses the TI post processing for values.



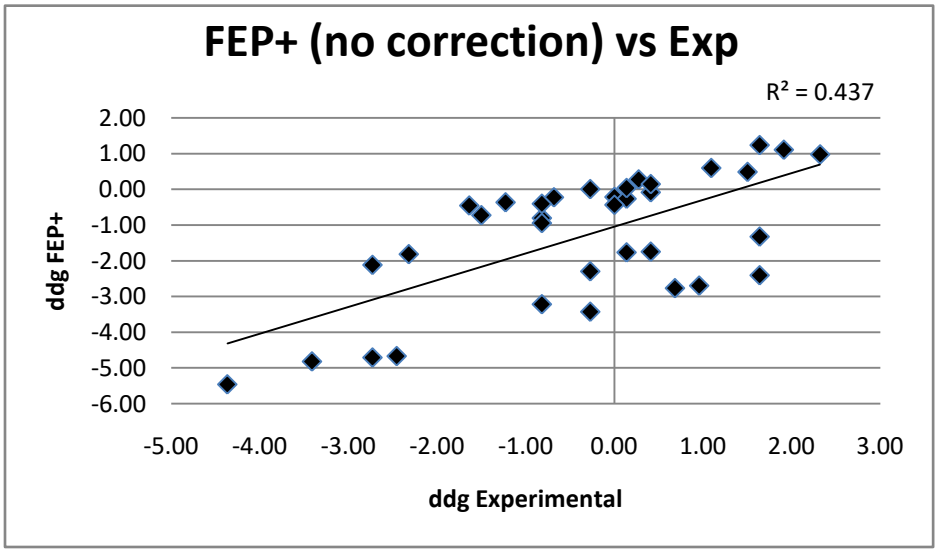


Figure 16: These figures show FEP+ results for individual edge values transformations for the TNNI3K system. Bottom shows the no correction raw output of the analysis, while top shows the result after FEP+'s cycle closure correction algorithm is applied.

To analyze the results and find which molecules would be chosen in a pharmaceutical standpoint, where only the top N molecules are selected, it is pertinent to find a metric that the values can be judged against. Therefore, it would be best to calculate down the edge map (Figure 18) for ddg values against the same starting molecule (Table 7). Due to not knowing whether sampling the entire Hamiltonian path would be more accurate due to cancellation of

errors or sampling less, sampling was carried out at several different intervals to determine the most accurate values. An assumption made when calculating was that the transformation are conducted from A->B, but the analysis program assumes B->A has the negative relative free binding energy of A->B. Instead of calculating all distances, it was more computationally sound to calculate only up to a maximum number of jumps. In order to obtain all the possible paths, a depth based search approach was used to find all possible paths from ligand A to ligand B.

Upon analysis of the values based on different pass counts (Table 4), it would appear that the more passes being calculated created an overall better correlation with experimental values for all methods. When using a minimum distance method, which was the minimum distance for each molecule to reach its goal from the origin molecule (all molecules used ligand 6 as the origin). If some molecule's paths took 2 edges away from the starting molecule to reach their end goal, and others took 3 edges, those that took 2 edges to reach their end goal would not use any values that reach the end goal in 3 edges. If there was more than one path for the minimum distance then the average of the paths energies was used. It appears that adding together all possible values in a distance rather than just taking the minimum distance gave better results for relative free energy. However, calculating too many paths became an exercise in propagating the inherent errors that already exist in the methods. For this system, the minimal distance away that covered all molecules was 3 away.

Table 4: Maximum number of paths from ligand 6 (refer to table 6 for ligand numbering) to target ligand vs Correlation with experiment (R^2)

Type	FEP+ (cycle closure)	FEP+ (no correction)	TI	MBAR
Minimum Path	0.765253	0.554247	0.627983	0.484755
3 Away	0.706909	0.697370	0.681839	0.657655
4 Away	0.710565	0.697370	0.669694	0.696755
5 Away	0.712226	0.691750	0.673669	0.685505
6 Away	0.710041	0.686627	0.674354	0.680368
7 Away	0.711387	0.692440	0.671917	0.686205
8 Away	0.711032	0.688152	0.674533	0.685795
All Possible Paths	0.694673	0.668739	0.697332	0.652754

Using 5 passes as it seems to be the beginning of the convergence of the values, it is possible to check how many molecules would make it past screening vs experimental values by checking to see if the top n molecules in experiment match with the top n molecules in a computational method (Table 5). When analyzing this, it seems that all 3 methods perform fairly well, all getting 80% of the top 5 molecules correct, albeit, in different orders. When using all possible paths to the end points, all three methods perform worse, only selecting 4/7 of the top molecules (Table 6).

Table 5: Number of pharmaceutical compounds selected using ddG values obtained from passes within 5 away of ligand 6

Number selected	FEP+ (cycle closure)	FEP+ (no correction)	TI	MBAR
Top1	0	0	0	0
Top 2	0	0	0	0
Top 3	0	2	2	1
Top 4	3	3	3	3
Top 5	4	4	4	4
Top 6	4	5	5	4
Top 7	5	5	5	4

Table 6: Number of pharmaceutical compounds selected using all possible paths from ligand 6 to other ligands.

Number selected	FEP+ (cycle closure)	FEP+ (no correction)	TI	MBAR
Top1	0	0	0	0
Top 2	0	0	1	0
Top 3	2	1	2	2
Top 4	3	1	3	3
Top 5	3	2	3	3
Top 6	3	3	4	4
Top 7	4	4	4	4

SAMPL6 Challenge

The following is work done on the SAMPL6 challenge part of D3R[101] which ended up unpublished due to time constraints and insufficient results.

Absolute free energy can also be calculated using the Thermodynamic Integration method[102]. This is a far less popular approach as absolute free energies take longer to converge and there is generally a lack of a baseline to gauge the "correctness" of the results. To do so, a solvent and complex simulation can be run where the solvent simulation contains the ligand in solvent, while the complex contains the protein complex with the ligand in solvent. In both sets of simulation the two sets of softcores for 0.0 and 1.0 lambda windows are the ligand vs no ligand. This allows a sampling of phase spaces where the ligand is shown and not shown.

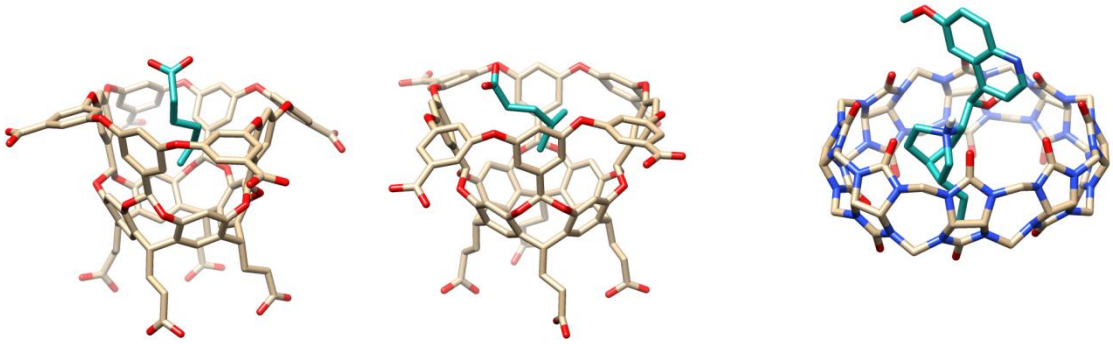


Figure 17: Left shows octaacid with Ligand 1, middle shows octaacid with Ligand 2, and right shows cucurbit[8]uril with Ligand 3.

The systems contained two host guests systems, octaacid with two ligands, as well as Cucurbit[8]uril (Figure 17). Ligands were provided with two for octacid and one for CB8 in 5 different orientations to test convergence on the system. Systems were run for 15 nanoseconds where the first 5 nanoseconds were disregarded from analysis, using 11 windows from 0.0 to 1.0 separated by 0.1. Amber TI was able to generate convergence on these systems (Table 7).

Table 7: dG values based on ligand done with several poses.

	Octaacid Ligand 1	Octaacid Ligand 2	Cucurbit[8]uril Ligand 3
Pose 1	-6.17	-10.76	-14.69
Pose 2	-7.89	-11.77	-15.35
Pose 3	-7.49	-10.73	-15.93
Pose 4	-6.51	-11.64	-10.32
Pose 5	-6.69	-11.05	-15.29
Experimental	-5.18	-6.22	-6.45

Although the Amber TI was able to reach relative convergence in these systems, the results varied from the results other methods gave. This was attributed to the differences in weights and cutoffs the Amber TI system used. Amber used a 10 angstrom cutoff, whereas other methods used a 9 angstrom cutoff with a van der waals force switch applied from 9 to 10 angstrom cutoff. Also, the restraints, used in Amber were generated using a virtual bond analysis [103], which were stronger than the restraints used in other methods. These restraints were needed as simulations that ran without the restraints connecting the ligand to the complex, the ligand would slip out of the binding pocket within the first few nanoseconds.

Due to these minute changes that needed to be made to gain “accurate” absolute free energy calculations, these calculations seem to be very finicky and need to be hand tuned on a

per system basis to obtain accurate free energy calculations. This is likely due to the fact that absolute free energy methods have more degrees of freedom in movement as they are not constrained by a linear region that is commonly seen in relative free energy calculations. It may also take significantly more statistical sampling for these calculations to reach their completion [104, 105]. This becomes very difficult to scale up to a larger setting, and make an all-encompassing method of absolute free energy methods using thermodynamic integration still a distant dream especially given the difficulty to validate free energies of new compounds without a reference.

Table 8: Individual ddG values with their transformation ligands. Energies in kcal/mol

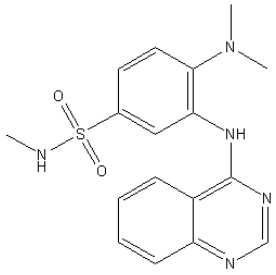
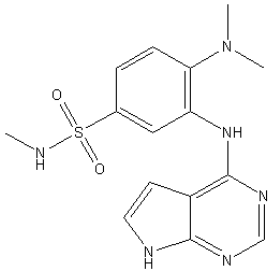
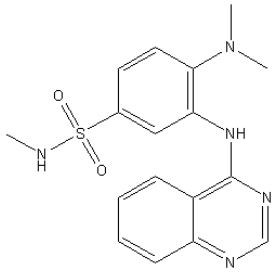
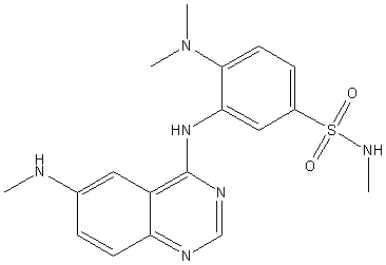
Ligand Before	Ligand After	ddG Exp.	ddG AMBER	ddG FEP+	ddG MBAR
		**-0.82**	**-1.69**	**-0.66**	**-2.09**
		**-0.82**	**-0.99**	**-0.86**	**-1.29**

Table 8 (continued): Individual ddG values with their transformation ligands. Energies in kcal/mol

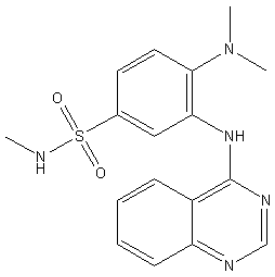
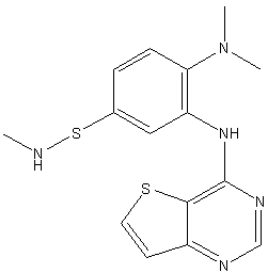
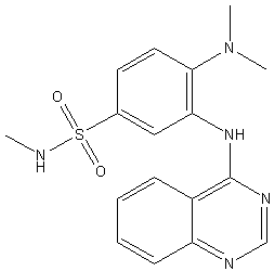
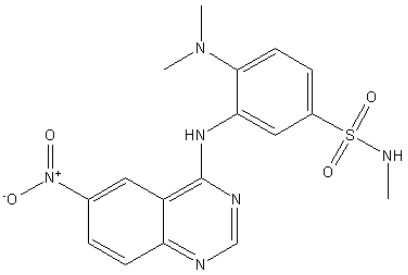
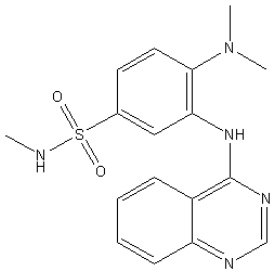
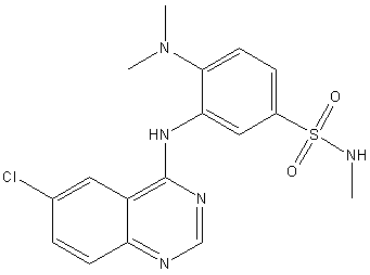
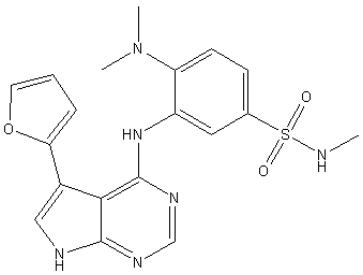
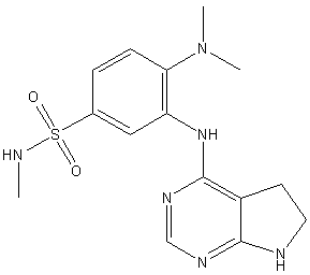
Ligand Before	Ligand After	ddG Exp.	ddG AMBER	ddG FEP+	ddG MBAR
		**0.00**	**-0.88**	**-0.32**	**-0.39**
		**1.50**	**0.47**	**0.41**	**-0.15**
		**0.41**	**-0.48**	**-0.15**	**0.79**
		**0.14**	**-0.73**	**-0.29**	**-0.32**

Table 8 (continued): Individual ddG values with their transformation ligands. Energies in kcal/mol

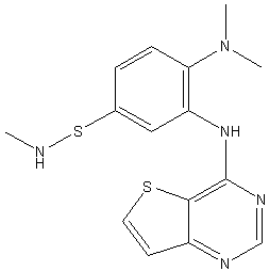
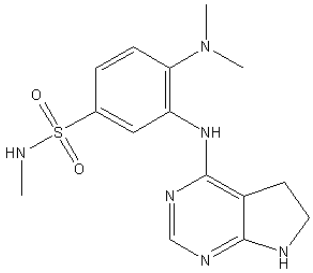
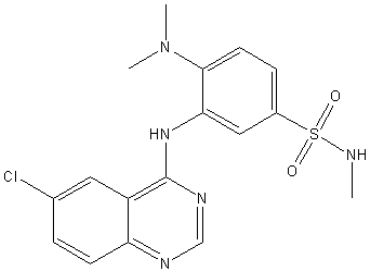
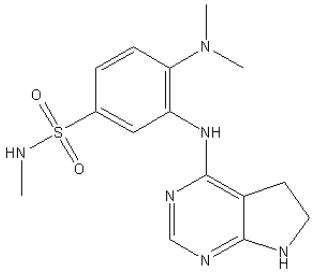
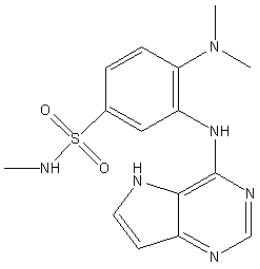
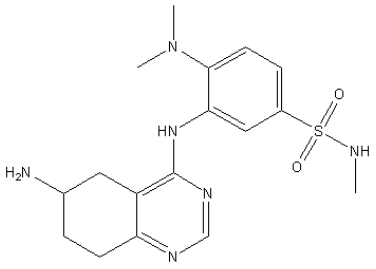
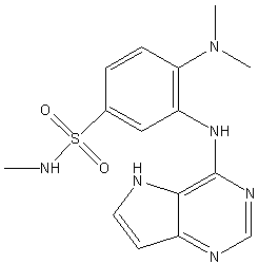
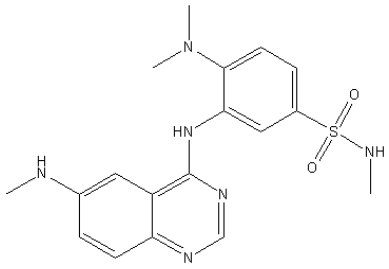
Ligand Before	Ligand After	ddG Exp.	ddG AMBER	ddG FEP+	ddG MBAR
		**-2.32**	**-2.41**	**-1.88**	**-2.66**
		**-2.73**	**-2.83**	**-2.05**	**-2.33**
		**0.14**	**-1.46**	**-1.76**	**-2.93**
		**0.41**	**-1.55**	**-1.99**	**-1.14**

Table 8 (continued): Individual ddG values with their transformation ligands. Energies in kcal/mol

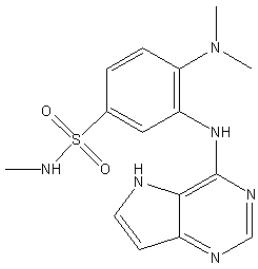
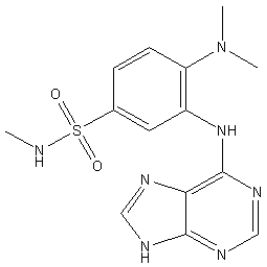
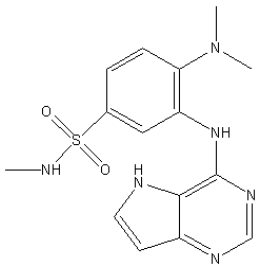
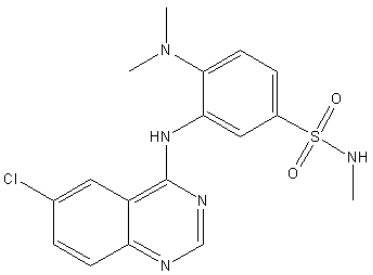
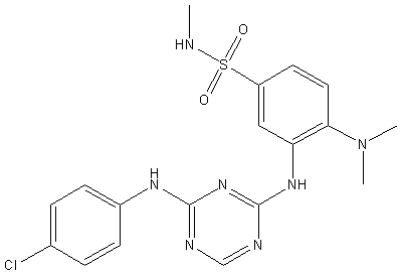
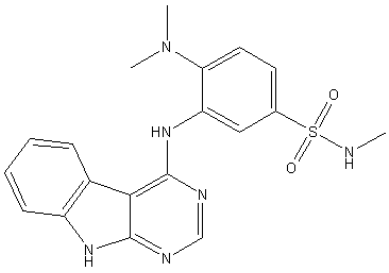
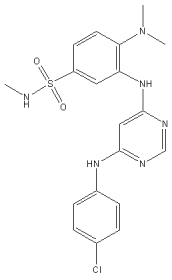
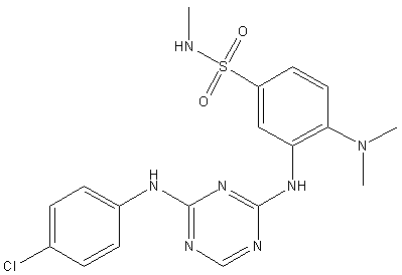
Ligand Before	Ligand After	ddG Exp.	ddG AMBER	ddG FEP+	ddG MBAR
		**-0.27**	**-2.87**	**-2.12**	**-2.03**
		**1.64**	**-1.62**	**-1.28**	**-1.80**
		**0.96**	**0.77**	**-2.82**	**-0.68**
		**-1.23**	**-2.82**	**-0.49**	**-3.89**

Table 8 (continued): Individual ddG values with their transformation ligands. Energies in kcal/mol

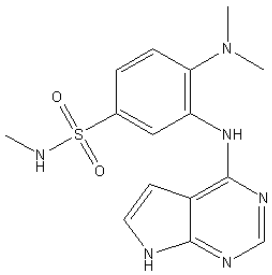
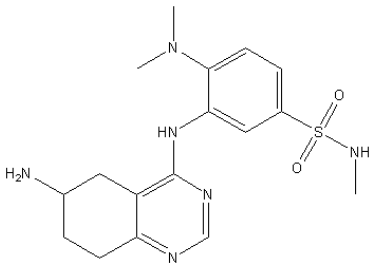
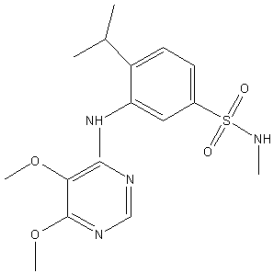
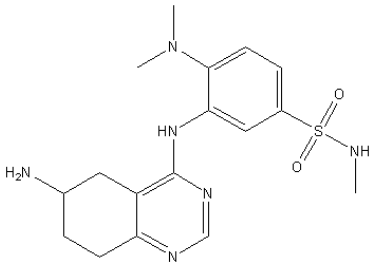
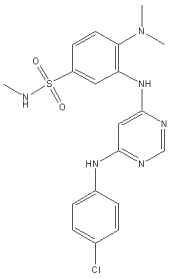
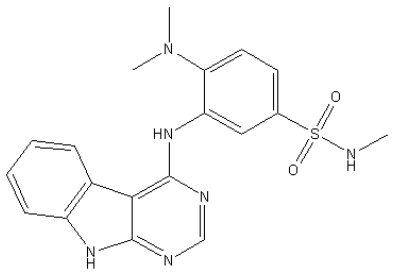
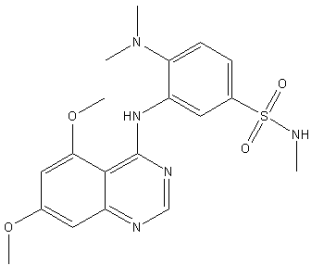
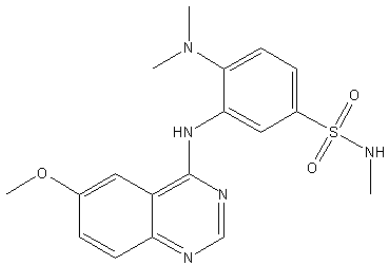
Ligand Before	Ligand After	ddG Exp.	ddG AMBER	ddG FEP+	ddG MBAR
		**-0.27**	**0.92**	**0.02**	**0.43**
		**0.14**	**0.64**	**0.00**	**1.03**
		**-0.27**	**-1.42**	**-3.31**	**-1.52**
		**1.91**	**1.73**	**0.93**	**0.84**

Table 8 (continued): Individual ddG values with their transformation ligands. Energies in kcal/mol

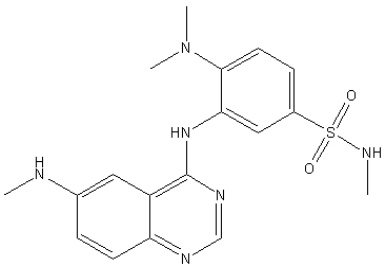
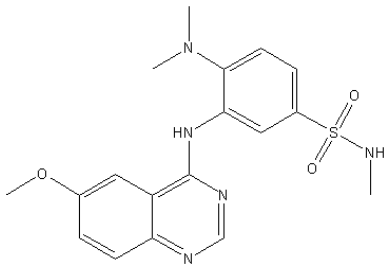
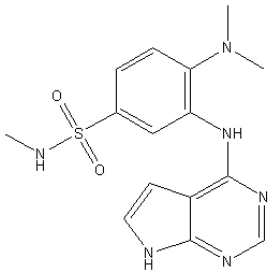
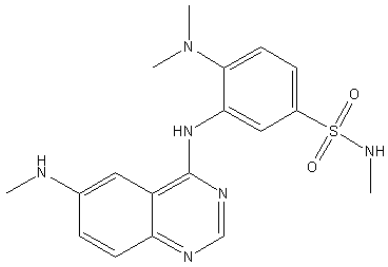
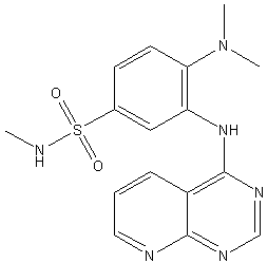
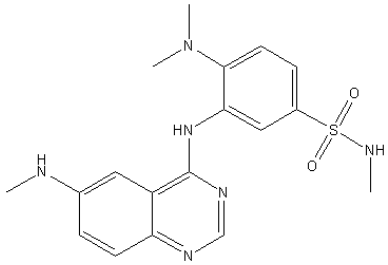
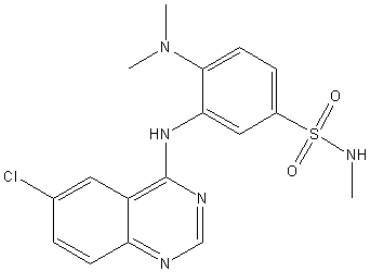
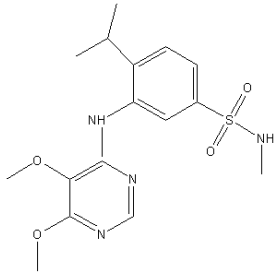
Ligand Before	Ligand After	ddG Exp.	ddG AMBER	ddG FEP+	ddG MBAR
		**0.27**	**0.27**	**0.28**	**1.00**
		**0.00**	**1.08**	**-0.21**	**0.49**
		**-2.73**	**-3.33**	**-4.79**	**-3.50**
		**-1.64**	**-1.20**	**-0.49**	**0.04**

Table 8 (continued): Individual ddG values with their transformation ligands. Energies in kcal/mol

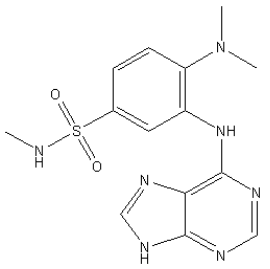
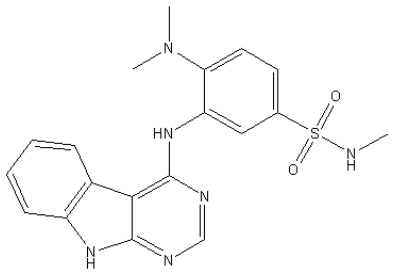
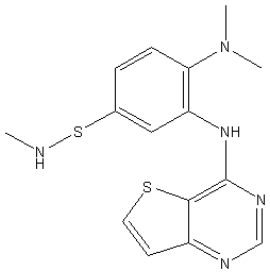
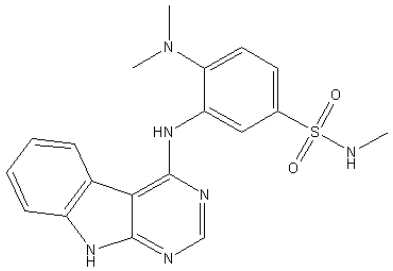
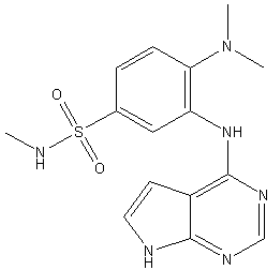
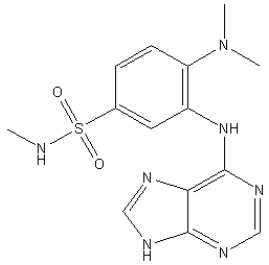
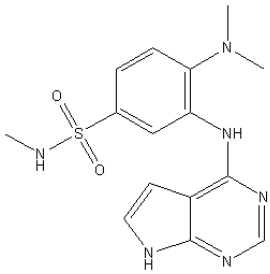
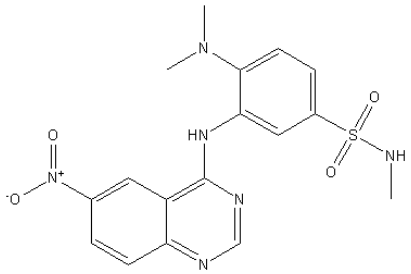
Ligand Before	Ligand After	ddG Exp.	ddG AMBER	ddG FEP+	ddG MBAR
		**0.68**	**-1.99**	**-2.75**	**-2.10**
		**-0.82**	**-2.53**	**-3.42**	**-3.33**
		**-0.68**	**-0.37**	**-0.34**	**0.42**
		**2.32**	**1.17**	**1.07**	**0.54**

Table 8 (continued): Individual ddG values with their transformation ligands. Energies in kcal/mol

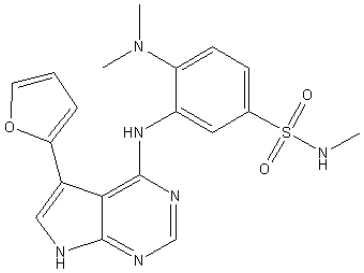
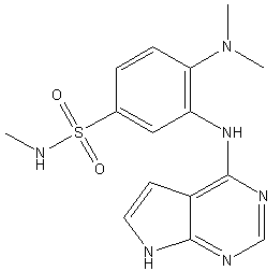
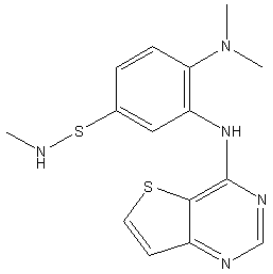
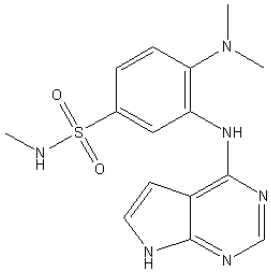
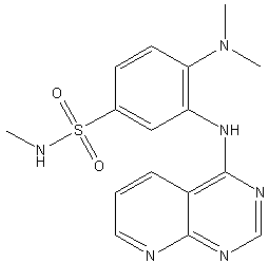
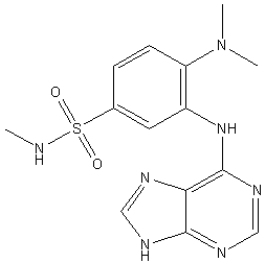
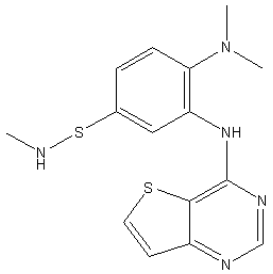
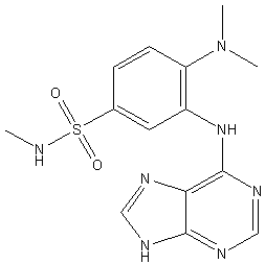
Ligand Before	Ligand After	ddG Exp.	ddG AMBER	ddG FEP+	ddG MBAR
		**1.64**	**0.94**	**1.25**	**0.84**
		**-0.82**	**-0.80**	**-0.34**	**-0.64**
		**-3.41**	**-4.42**	**-4.92**	**-4.43**
		**-1.50**	**-0.92**	**-0.68**	**-0.03**

Table 8 (continued): Individual ddG values with their transformation ligands. Energies in kcal/mol

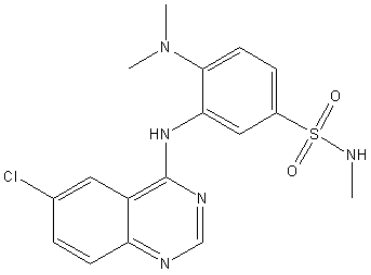
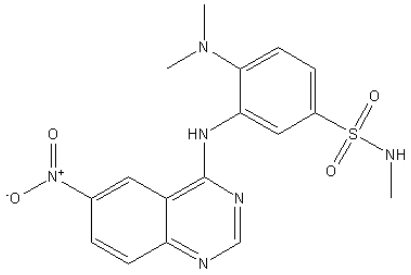
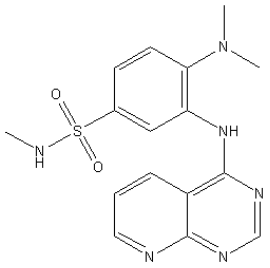
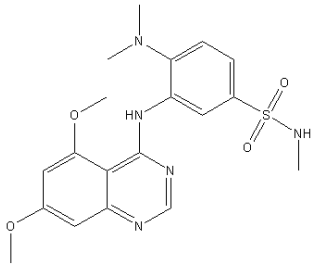
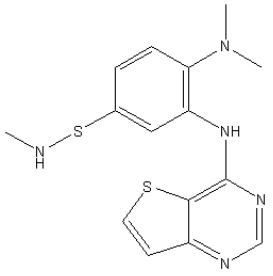
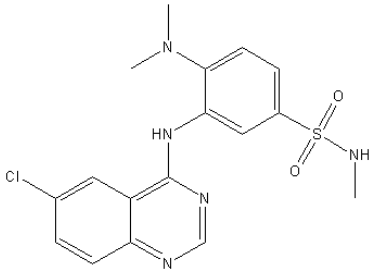
Ligand Before	Ligand After	ddG Exp.	ddG AMBER	ddG FEP+	ddG MBAR
		**1.09**	**0.43**	**0.56**	**0.78**
		**-4.37**	**-5.78**	**-5.44**	**-6.52**
		**0.41**	**0.62**	**0.17**	**1.07**

Table 9: ddG values of ligands using Ligand 6 as the starting using all paths. Energies in kcal/mol.

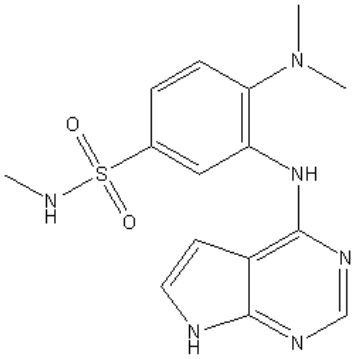
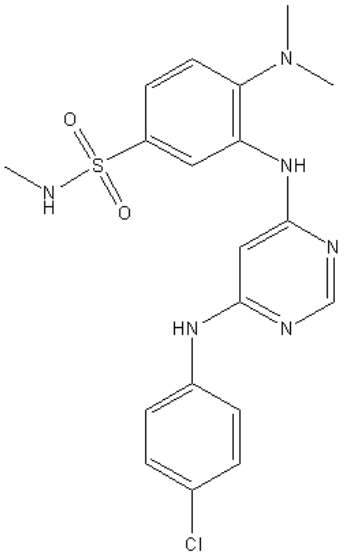
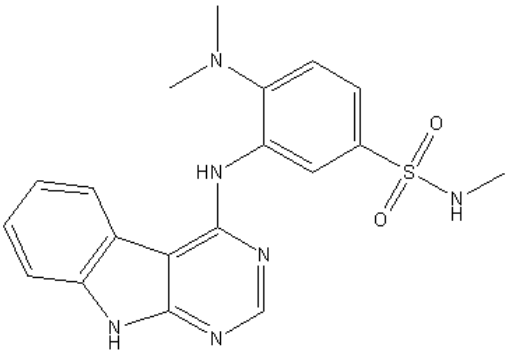
ID	Molecule	Exp	FEP+ (correction)	TI	MBAR
1		-0.82	-1.56583	-0.7862	-1.35815
2		-0.55	-2.1644	-0.5312	-1.4984
3		-0.82	-3.6595	-3.8515	-3.75083

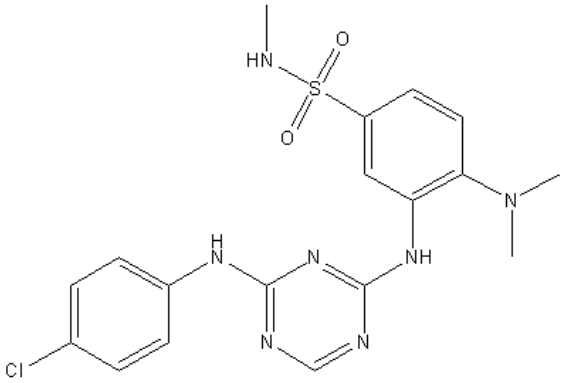
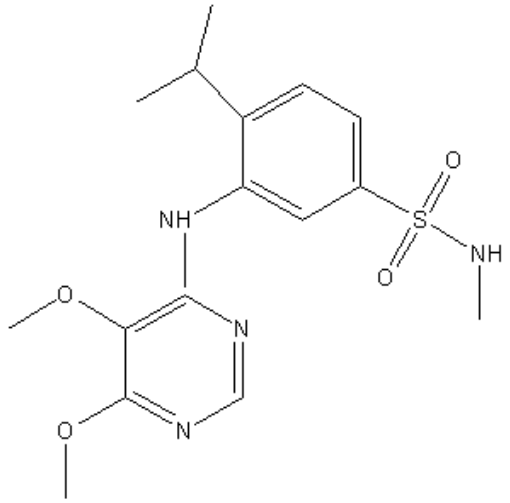
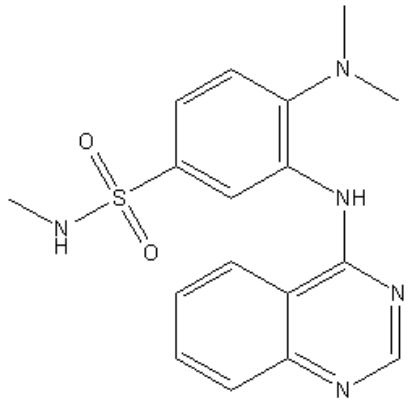
Table 9 (continued): ddG values of ligands using Ligand 6 as the starting using all paths. Energies in kcal/mol.					
ID	Molecule	Exp	FEP+ (correction)	TI	MBAR
4		-1.78	-4.6064	-1.0212	-3.5584
5		-1.23	-1.5265	-0.72125	-0.93125
6		0	0	0	0

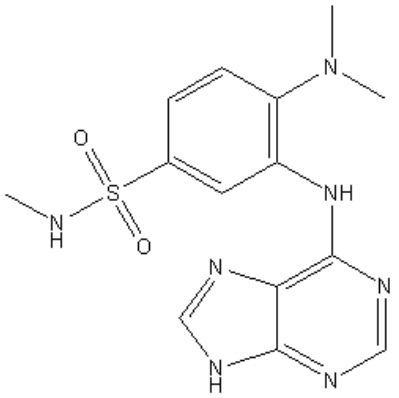
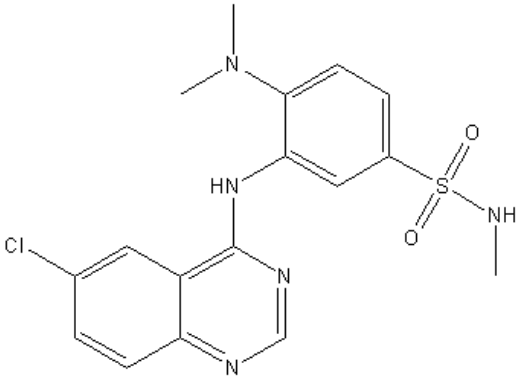
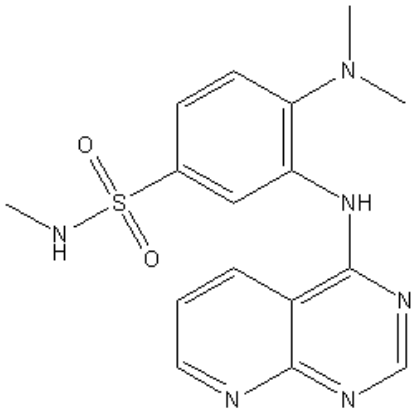
Table 9 (continued): ddG values of ligands using Ligand 6 as the starting using all paths. Energies in kcal/mol.					
ID	Molecule	Exp	FEP+ (correction)	TI	MBAR
7		-1.5	-1.76054	-1.11811	-0.82135
8		0.41	-0.40528	-0.30697	-0.18225
9		1.91	2.82397	3.82621	3.54983

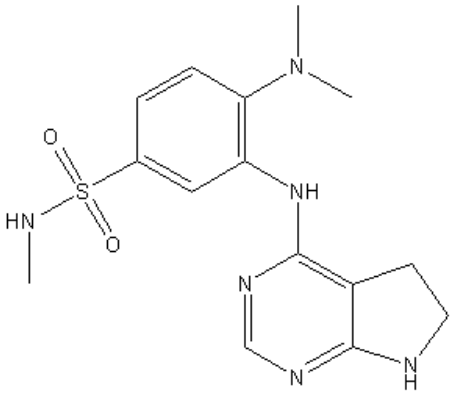
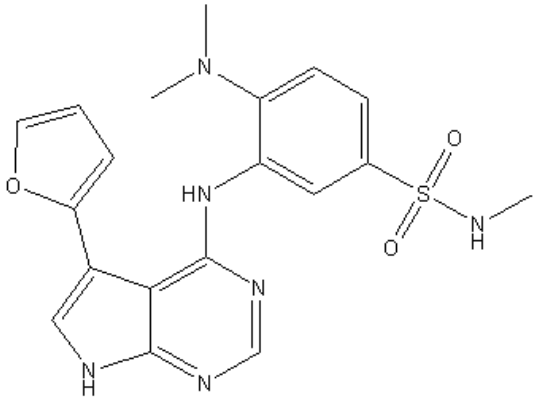
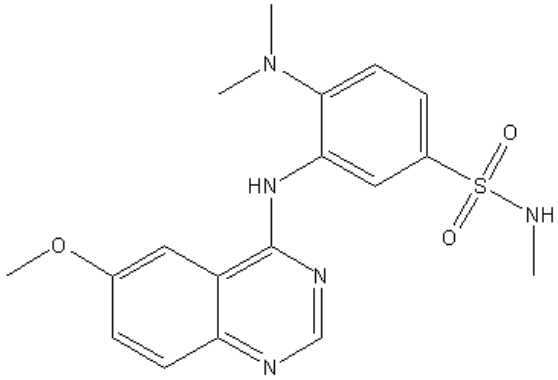
Table 9 (continued): ddG values of ligands using Ligand 6 as the starting using all paths. Energies in kcal/mol.					
ID	Molecule	Exp	FEP+ (correction)	TI	MBAR
10		-2.32	-3.34255	-2.28891	-2.97927
11		-2.46	-2.51921	-2.01553	-2.33868
12		-0.55	-0.7148	-0.6506	-0.0368

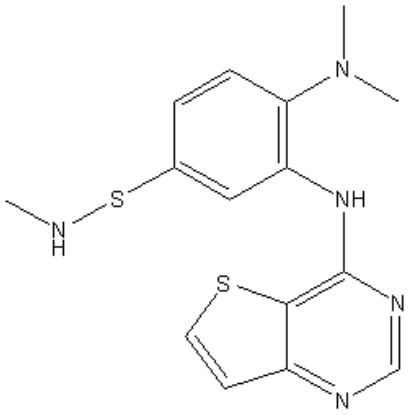
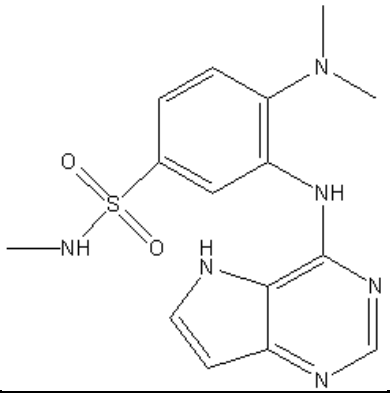
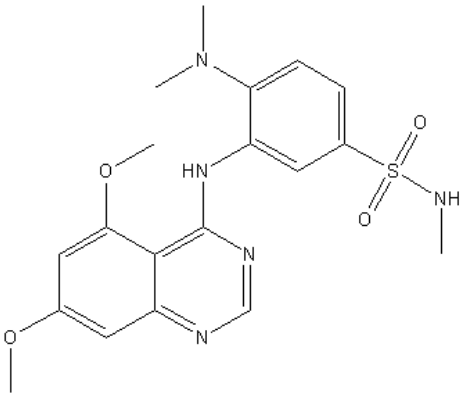
Table 9 (continued): ddG values of ligands using Ligand 6 as the starting using all paths. Energies in kcal/mol.					
ID	Molecule	Exp	FEP+ (correction)	TI	MBAR
13		0	-0.9874	-0.37875	-0.74458
14		-1.23	0.993714	1.03243	1.31143
15		-2.46	-2.67614	-1.62659	-1.925

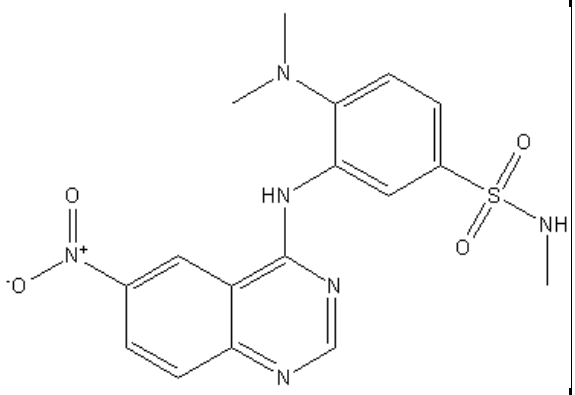
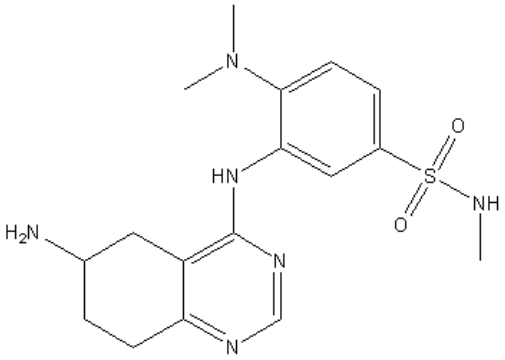
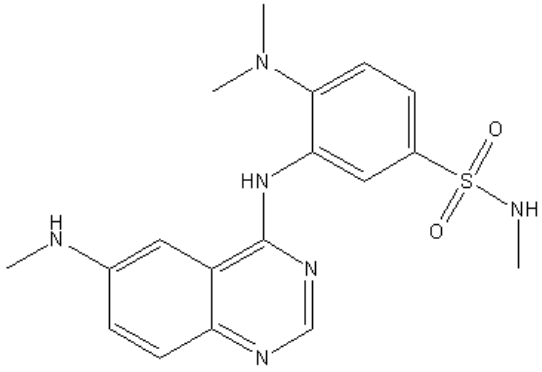
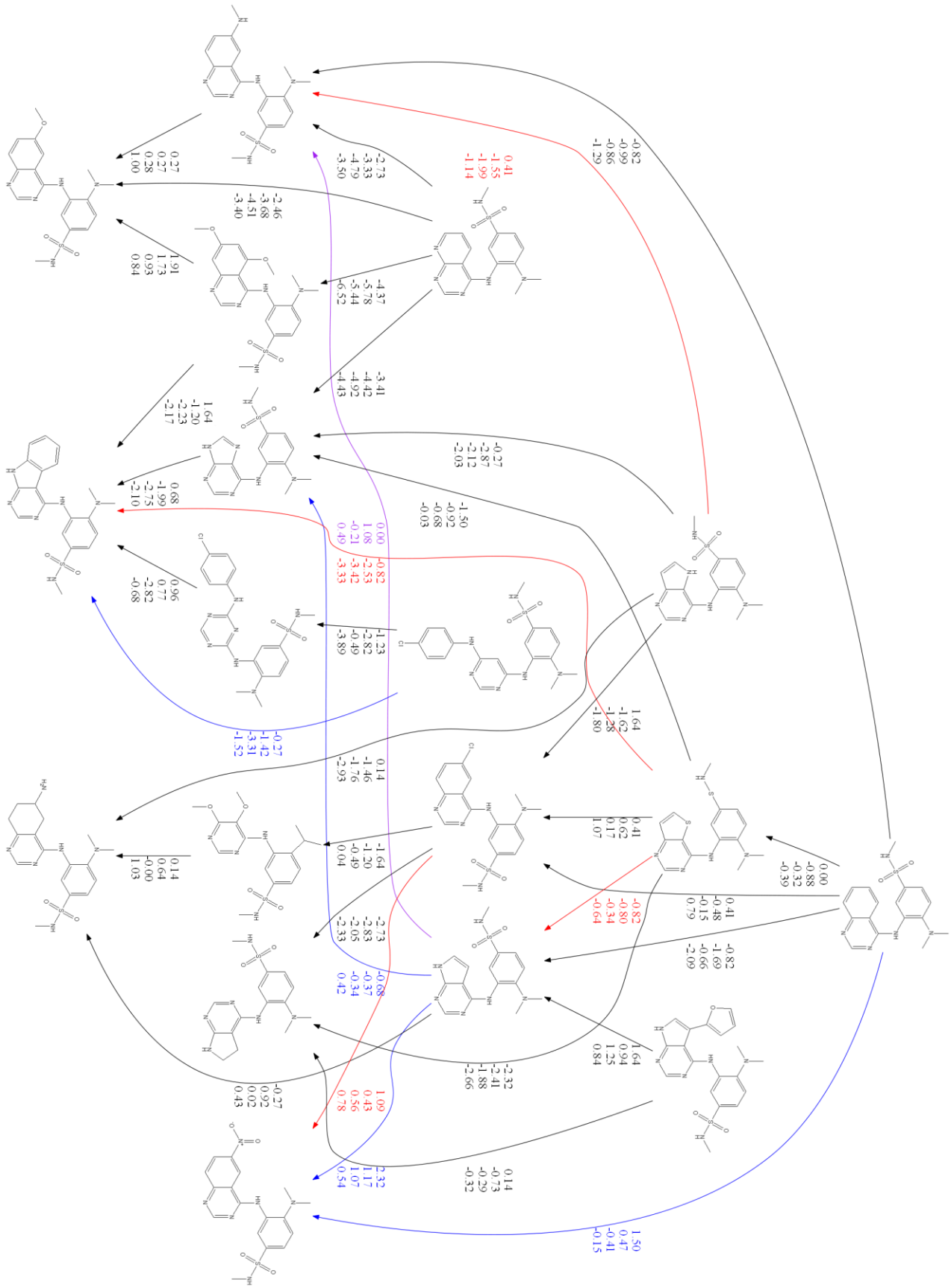
Table 9 (continued): ddG values of ligands using Ligand 6 as the starting using all paths. Energies in kcal/mol.					
ID	Molecule	Exp	FEP+ (correction)	TI	MBAR
16		1.5	-0.05409	0.149848	-0.05712
17		-1.09	-0.5918	-0.7352	-1.0606
18		-0.82	-0.69558	-0.92174	-0.56954

Figure 18 (Next Page): Edge graph of TNNI3K system. Numbers are the ddg value from changing from one compound to another with the first number being the experimental value followed by AmberTI, followed by FEP+ with cycle closure correction, and lastly with MBAR values.



Chapter 5: Additional Methodologies for Molecular Dynamics

With the ability to do fast and complex simulations in a reasonable timeframe, it becomes more feasible to develop, implement, and use novel complex methods to better sample and mimic biological structure and function. Beyond being able to run long timescale simulations, several methods are implemented to try to increase the sampling in Molecular Dynamics and simulate these long timescale simulations. Methods such as Accelerated Molecular Dynamics [106] and Gaussian Accelerated Molecular Dynamics [107] attempt to increase sampling by changing the molecular barriers in a system such as torsional costs and overall potential energy barriers. Other methods such as replica exchange [108] allow the sampling of many temperatures or pH levels to better sample a system as the replicas exchange between each other to allow conformations from different environments to run in different environments hence the name. These methods do not replace the need of a faster molecular dynamics engine as they do not maintain proper dynamics that can be gathered from a pure molecular dynamics simulation. One of the major ideals in molecular dynamics is the ability to simulate a complex membrane environment with ion channels and passive diffusion. In order to accomplish this, initial work was carried out as part of this thesis on the ability to accurately build lipid membranes on all lipid forcefields, passive diffusion, and electric fields. The idea for ionic gradients was to use asymmetric periodic boundary conditions [109] (Figure 19), but due to code implementation details in Amber 16, the idea was deemed insufficient and shelved while changes in the code with the midpoint code fixed these obstacles.

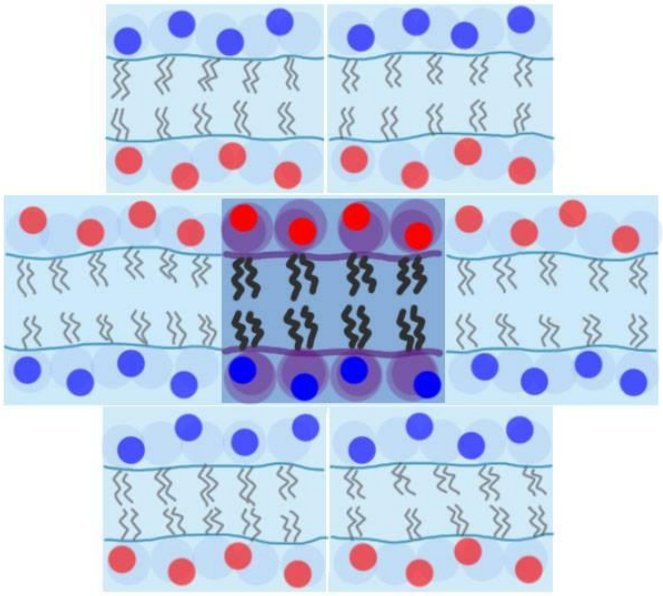


Figure 19: An ideal periodic box setup that would allow for ionic gradiants. Alternate layers of boxes are offset by 1/2 to allow much like a brick structure. Red and blue represent different ions.

In addition, work was carried out to help better find water pockets in proteins and gain better absolute free energies with the Monte Carlo water code.

Force Switch

The Lipid 14 force field [32] was released by the Walker group with the Amber 16 release using a parameterization method that allowed for explicit parameterization of the lipid head groups and lipid tail groups. This allowed for the users to pick which lipid head and tail group combinations they needed for their simulation. The lipid forcefield was validated using several experimental parameters such as lipid per area of the lipid membrane and lipid bilayer thickness. However, perhaps the most impressive of the validation methods was the ability for the lipid bilayer to self-assemble over many nanoseconds period[110]. The Lipid 14 forcefield, and its recent update, Lipid 17, is one of many all-atom lipid forcefields. Other include CHARMM 36[111-113] and SLipids[114, 115].

As described in Chapter 1, AMBER uses a cut off method that does not compute forces past a cut off point. When attempting to self-assemble DPPC lipids using the CHARMM 36 forcefield without force switching resulted in a gel phase in the lipid bilayer. (Figure 20)

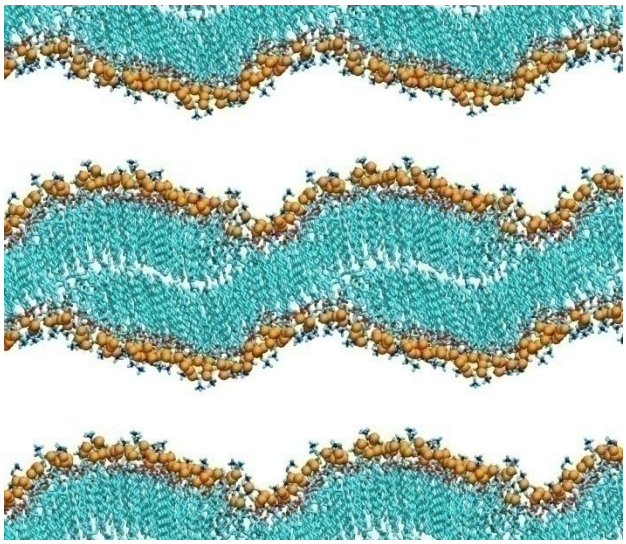


Figure 20: Simulations of hydrated DPPC bilayer using CHARMM v36 forcefield in AMBER without force switching creates a gel phase. Average area/lipid = 53.8 Å^2.

However, when applying a force switch[116], where the forces augment to 0 past a cutoff:

$r<r_{on}$

$$V(r) = V_{true}(r) + \frac{-c}{r_{on} * r_{off}}$$

$r_{on}<r<r_{off}$

$$V(r) = \frac{c * {r_{off}}^{b}}{{r_{off}}^{b} - {r_{on}}^{b}} (r^{-b} - {r_{off}}^{-b})^2 \quad (10)$$

the DPPC lipids self-assemble from a liquid disorder phase. Phospholipid tails are largely hydrophobic and in the case of some force field parameterizations (for example DPPC in Charmm c36) can be overly sensitive to the treatment of the VDW cutoff. Subtle cancellation of errors in the parameter fitting process means that in these cases it is necessary to run with identical settings to the parameterization in order to recreate these cancellations of error and give the desired behavior such as that of DPPC (Figure 19).

Due to the simulation demands of self-assembly, being about 10 microseconds of simulation time, these simulations are too computational expensive to perform on CPU and therefore, force switching was needed to be implemented on the GPU to perform these simulations. Thus, force switching was implemented and added in AMBER 16. By utilizing force switching on a DPPC bilayer using the CHARMM v36 forcefield, a liquid disorder phase, similar to that of human structure can be obtained (Figure 21).

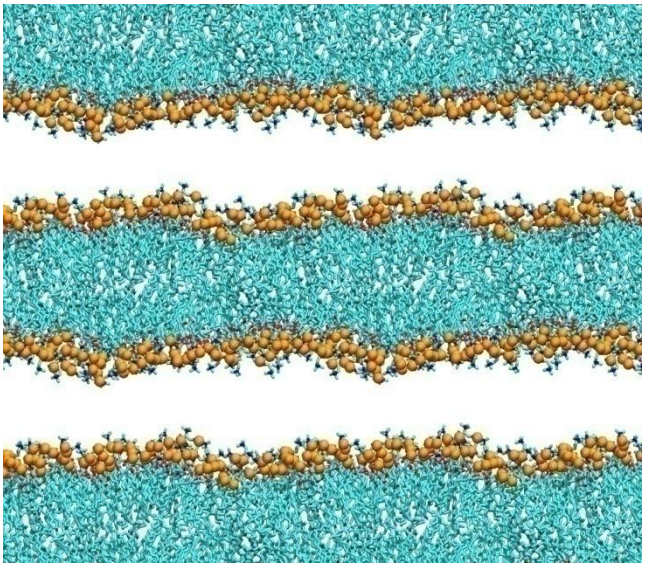


Figure 21: Simulations of hydrated DPPC bilayer using CHARMM v36 forcefield in AMBER with force switching. Adding this method creates a liquid disorder phase representative of human systems. Average area/lipid = 62.3 Å^2. Experimental value is observed to be 63.1-64.3 Å ^2

Directional Restraints

The following was work done in conjunction with Callum Dickson [117] to simulate diffusion through a membrane. The best way to do this in Molecular Dynamics is the idea of umbrella sampling. This is a sample of phase space that a molecule can explore as it is manually pulled through the membrane. To most accurately do this, keeping a molecule a distance away from the lipid membrane, and doing simulations that progress closer and closer until it goes through the membrane is the ideal way to explore phase space. However, as the molecule needs

to explore at a certain z-distance away from the lipid bilayer, it is pertinent to have a method that keeps the molecule in a certain z level[118, 119]. For this purpose, directional restraints were implemented in AMBER 16.

Using, AMBER's general restraints, the distance between two center of masses is calculated using the standard distance formula:

$$r = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2 + (z_i - z_j)^2}. \quad (11)$$

Assume an energy potential is defined by distances r_1, r_2, r_3, and r_4 where the distance between r_2 and r_3 is the optimal distance between the two center of masses. The energy potential from r_1 to r_2 and r_3 to r_4 are parabolic. From r_2 to r_3 the energy potential is flat. Behind r_1 and behind r_4 the potential is linear. Therefore, the equations AMBER uses for its restraints are:

$r<r_1$

$$E = 2 * k_2 * (r_1 - r_2) * (r - r_1) + k_2 * (r_1 - r_2)^2$$

$$df = 2 * k_2 * (r_1 - r_2)$$

$r_1<r<r_2$

$$E = k_2 * (r - r_2)^2$$

$$df = 2 * k_2 * (r - r_2)$$

$r_2<r<r_3$

$$E = 0$$

$$df = 0$$

$r_3<r<r_4$

$$E = k_3 * (r - r_3)^2$$

$$df = 2 * k_3 * (r - r_3)$$

$r>r_4$

$$E = 2 * k_3 * (r_4 - r_3) * (r - r_4) + k_3 * (r_4 - r_3)^2$$

$$df = 2 * k_3 * (r_4 - r_3)$$

For directional restraints, what is desired is:

$$r = \sqrt{(w_x * d_x)^2 + (w_y * d_y)^2 + (w_z * d_z)^2} \qquad (12)$$

In this equation, w is the weight for the x, y, z direction, and d is the distance of the x, y, z axis of the center of mass. When the weight for a direction is set to 0, then a restraint is not evaluated in that axis. This allows for the effects of the restraints to only be applied in one direction rather than all directions. By implementing directional restraints by simply augmenting the distance formula, the energy equation is not affected.

Additionally, as the lipid bilayer is usually centered at z=0, a molecule that is held to place at z=0, using a distance formula to keep track of the center of mass distance, it becomes important to output a vector for the distance rather than the absolute distance. This feature was only enabled for single direction restraints.

Electric Fields

In order to simulate ion channels, an electric field is necessary. A simple external electric field that modulates the system forces by $E = e_{i,j} * q_{i,j}$, was implemented in AMBER 16, where i is the index of the atom and j is the x, y, or z component, where e is the gradient of the electric field and q is the charge of the system. Difficulty arises when considering the energy or virial. Given the external nature of this force, momentum is not conserved as it is with bonds, angles, dihedrals, and non-bonded forces (which are also dealt with using periodic nature of our system box). Therefore, it is not possible to deal with a virial and thus the electric field was only implemented in NVT. This provides significant constraints as the lateral area of the lipid bilayer

would not be able to relax. It is therefore in the best interest to run large NVT bilayer systems with well-equilibrated lipid bilayers to compensate for the lack of barostat.

Monte Carlo Water

In order to detect hidden water pockets and more accurately solvate systems for free energy calculations, a Metropolis Monte Carlo water method was implemented. The core concept of the idea was to randomly displace water molecules and use acceptance criteria of

$$e^{-\frac{E_0-E_1}{KT}} > R \quad (15)$$

where R is a random number between 0 and 1, and E0, E1 represent the old and new energies to determine the new position of the water. If the acceptance criteria energy passes, the new state is taken, if failed, revert to the old state. By using Metropolis Monte Carlo acceptance criteria, if a state changes the energy adversely its acceptance chance is almost zero resulting in an overall stable simulation. This method, the Monte Carlo water method, was developed and released for Amber 18.

When developing this method, several key factors were analyzed and fixed. One of the problems being how random is the amber random number generator. The issue with the random number generator is that it creates a Gaussian distribution of random numbers, but in doing so, it has issues where it favors whole fractions such as 0, 1/2, 1/3, ¼, etc. This causes 0 to occur more often and with 0 being the lowest number in the Monte Carlo acceptance criteria, this allows some very unfavorable conformations to be selected. To fix this, the intrinsic Fortran random number generator is used for the Monte Carlo water simulations.

Optimizations and continual development of the code is continuing in collaboration with Ido Ben-Shalom of the Gilson group at UCSD, in particular focusing on ways to accelerate the

energy cost of trial moves which is especially expensive. This cost is primarily due to the displacement of the water molecules making the previous neighbor list invalid. This created a need to rebuild the entire neighbor list each time the Monte Carlo code was called. To remedy this, optimizations were created to ensure that water molecules cannot be in a position where they clash with one another using a steric grid that rejected moves where molecules would clash. Additionally, a coarse energy calculation was added that quickly computed the local area of change when a water molecule was displaced. This avoids the incredibly expensive full molecular dynamics call. Further details about these optimizations as well as the full details of implementation and applications will come in a future publication.

The idea behind the Monte Carlo method is that as proteins move and breathe, hidden pockets might arise that would usually already be solvated. Additionally when ligands leave binding pockets, this method can help fill the pocket with water as since in absolute free energy calculations the ligand existing and not existing occupy the V_0 and V_1 states of the simulation. By having water occupy the cavity as the ligand interactions disappear, it becomes a more realistic approach to free energy calculations.

Conclusion

By taking advantage of the many core systems, GPUs, and algorithmic improvements to performance, it becomes ever more feasible and better to run complex calculations and complex systems. It is no longer the vanilla molecular dynamics that is interesting in science, but the advanced sampling of phase space and the ability to do realistic all-atom simulations of complex molecular pathways and systems that are of great interest. If molecular dynamics is to be a huge part of the future, it will need to be able to accurately predict drug effects on pathways and

accurately predict in silico drugs for in vivo testing. Each new algorithm and every bit of performance helps slowly and surely brings molecular dynamics towards that future.

Chapter 5, in part, is a reprint of the material as it appears in Simulation of lipid bilayer self-assembly using all-atom lipid force fields in Physical Chemistry Chemistry Physics 2016. Shjevik Age A.; Madej, Benjamin D.; Dickson, Callum. J.; Lin, Charles; Teigen, Knut; Walker, Ross C., 2016. The dissertation author was an author of this paper.

References

1. Conway, S., Joseph, Earl C., Sorensen, Robert, *IDC: Quantum Computing in the Real World*. 2016: www.idc.com.

2. Nakamoto, S., *Bitcoin: A peer-to-peer electronic cash system.*

3. Wood, G., *ETHEREUM: A SECURE DECENTRALISED GENERALISED TRANSACTION LEDGER.* 2014.

4. Xu, D., M.J. Williamson, and R.C. Walker, *Chapter 1 - Advancements in Molecular Dynamics Simulations of Biomolecules on Graphical Processing Units*, in *Annual Reports in Computational Chemistry*, R.A. Wheeler, Editor. 2010, Elsevier. p. 2-19.

5. Götz, A.W., T. Wölfle, and R.C. Walker, *Chapter 2 - Quantum Chemistry on Graphics Processing Units*, in *Annual Reports in Computational Chemistry*, R.A. Wheeler, Editor. 2010, Elsevier. p. 21-35.

6. Allen, M.P.a.T., D.J., *Computer Simulation of Liquids*. 1987: Oxford Scienc.

7. Zijun, Y., *General thermal wavelength and its applications.* European Journal of Physics, 2000. **21**(6): p. 625.

8. *On the determination of molecular fields. —II. From the equation of state of a gas.* Proceedings of the Royal Society of London. Series A, 1924. **106**(738): p. 463.

9. Eastman, P., J. Swails, J.D. Chodera, R.T. McGibbon, Y. Zhao, K.A. Beauchamp, L.-P. Wang, A.C. Simmonett, M.P. Harrigan, C.D. Stern, R.P. Wiewiora, B.R. Brooks, and V.S. Pande, *OpenMM 7: Rapid development of high performance algorithms for molecular dynamics.* PLOS Computational Biology, 2017. **13**(7): p. e1005659.

10. Lindahl, E., B. Hess, and D. van der Spoel, *GROMACS 3.0: A package for molecular simulation and trajectory analysis*. Vol. 7. 2001. 306-317.

11. Hess, B., C. Kutzner, D. van der Spoel, and E. Lindahl, *GROMACS 4: Algorithms for Highly Efficient, Load-Balanced, and Scalable Molecular Simulation.* Journal of Chemical Theory and Computation, 2008. **4**(3): p. 435-447.

12. Pronk, S., S. Páll, R. Schulz, P. Larsson, P. Bjelkmar, R. Apostolov, M.R. Shirts, J.C. Smith, P.M. Kasson, D. van der Spoel, B. Hess, and E. Lindahl, *GROMACS 4.5: a high-throughput and highly parallel open source molecular simulation toolkit.* Bioinformatics, 2013. **29**(7): p. 845-854.

13. Abraham, M.J., T. Murtola, R. Schulz, S. Páll, J.C. Smith, B. Hess, and E. Lindahl, *GROMACS: High performance molecular simulations through multi-level parallelism from laptops to supercomputers.* SoftwareX, 2015. **1-2**: p. 19-25.

14. Brooks Bernard, R., E. Bruccoleri Robert, D. Olafson Barry, J. States David, S. Swaminathan, and M. Karplus, *CHARMM: A program for macromolecular energy, minimization, and dynamics calculations.* Journal of Computational Chemistry, 1983. **4**(2): p. 187-217.

15. Brooks, B.R., C.L. Brooks, A.D. MacKerell, L. Nilsson, R.J. Petrella, B. Roux, Y. Won, G. Archontis, C. Bartels, S. Boresch, A. Caflisch, L. Caves, Q. Cui, A.R. Dinner, M. Feig, S. Fischer, J. Gao, M. Hodoscek, W. Im, K. Kuczera, T. Lazaridis, J. Ma, V. Ovchinnikov, E. Paci, R.W. Pastor, C.B. Post, J.Z. Pu, M. Schaefer, B. Tidor, R.M. Venable, H.L. Woodcock, X. Wu, W. Yang, D.M. York, and M. Karplus, *CHARMM: The Biomolecular Simulation Program.* Journal of computational chemistry, 2009. **30**(10): p. 1545-1614.

16. Phillips James, C., R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, D. Skeel Robert, L. Kalé, and K. Schulten, *Scalable molecular dynamics with NAMD.* Journal of Computational Chemistry, 2005. **26**(16): p. 1781-1802.

17. Plimpton, S., *Fast Parallel Algorithms for Short-Range Molecular Dynamics.* Journal of Computational Physics, 1995. **117**(1): p. 1-19.

18. Case, D.A., T.E. Cheatham, T.O.M. Darden, H. Gohlke, R.A.Y. Luo, K.M. Merz, A. Onufriev, C. Simmerling, B. Wang, and R.J. Woods, *The Amber Biomolecular Simulation Programs.* Journal of computational chemistry, 2005. **26**(16): p. 1668-1688.

19. D.A. Case, I.Y.B.-S., S.R. Brozell, D.S. Cerutti, T.E. Cheatham, III, V.W.D. Cruzeiro, T.A. Darden, R.E. Duke, D. Ghoreishi, M.K. Gilson, H. Gohlke, A.W. Goetz, D. Greene, R Harris, N. Homeyer, S. Izadi, A. Kovalenko, T. Kurtzman, T.S. Lee, S. LeGrand, P. Li, C. Lin, J. Liu, T. Luchko, R. Luo, D.J. Mermelstein, K.M. Merz, Y. Miao, G. Monard, C. Nguyen, H. Nguyen, I. Omelyan, A. Onufriev, F. Pan, R. Qi, D.R. Roe, A. Roitberg, C. Sagui, S. Schott-Verdugo, J. Shen, C.L. Simmerling, J. Smith, R. Salomon-Ferrer, J. Swails, R.C. Walker, J. Wang, H. Wei, R.M. Wolf, X. Wu, L. Xiao, D.M. York and P.A. Kollman, *AMBER 2018*. 2018: University of California, San Francisco.

20. *The Amber Molecular Dynamics Software Suite*. 7/13/2018].

21. Wang, J., W. Wang, P.A. Kollman, and D.A. Case, *Automatic atom type and bond type perception in molecular mechanical calculations.* Journal of Molecular Graphics and Modelling, 2006. **25**(2): p. 247-260.

22. Christian E. A. F. Schafmeister, W.S.R.a.V.R., *LEaP*. 1995: University of California, San Francisco.

23. Cerutti, D.S. and D.A. Case, *Multi-Level Ewald: A hybrid multigrid / Fast Fourier Transform approach to the electrostatic particle—mesh problem.* Journal of chemical theory and computation, 2010. **6**(2): p. 443-458.

24. Roe, D.R. and T.E. Cheatham, *PTRAJ and CPPTRAJ: Software for Processing and Analysis of Molecular Dynamics Trajectory Data.* Journal of Chemical Theory and Computation, 2013. **9**(7): p. 3084-3095.

25. Shirts, M.R., C. Klein, J.M. Swails, J. Yin, M.K. Gilson, D.L. Mobley, D.A. Case, and E.D. Zhong, *Lessons learned from comparing molecular dynamics engines on the SAMPL5 dataset.* Journal of computer-aided molecular design, 2017. **31**(1): p. 147-161.

26. Betz, R. and R. C. Walker, *Paramfit: Automated Optimization of Force Field Parameters for Molecular Dynamics Simulations*. Vol. 36. 2015.

27. Duke, R.E., O.N. Starovoytov, J.-P. Piquemal, and G.A. Cisneros, *GEM*: A Molecular Electronic Density-Based Force Field for Molecular Dynamics Simulations.* Journal of Chemical Theory and Computation, 2014. **10**(4): p. 1361-1365.

28. Ren, P. and W. Ponder Jay, *Consistent treatment of inter- and intramolecular polarization in molecular mechanics calculations.* Journal of Computational Chemistry, 2002. **23**(16): p. 1497-1506.

29. Wang, J., M. Wolf Romain, W. Caldwell James, A. Kollman Peter, and A. Case David, *Development and testing of a general amber force field.* Journal of Computational Chemistry, 2004. **25**(9): p. 1157-1174.

30. Maier, J.A., C. Martinez, K. Kasavajhala, L. Wickstrom, K.E. Hauser, and C. Simmerling, *ff14SB: Improving the Accuracy of Protein Side Chain and Backbone Parameters from ff99SB.* Journal of Chemical Theory and Computation, 2015. **11**(8): p. 3696-3713.

31. Skjevik, Å.A., B.D. Madej, R.C. Walker, and K. Teigen, *LIPID11: A Modular Framework for Lipid Simulations Using Amber.* The Journal of Physical Chemistry B, 2012. **116**(36): p. 11124-11136.

32. Dickson, C.J., B.D. Madej, Å.A. Skjevik, R.M. Betz, K. Teigen, I.R. Gould, and R.C. Walker, *Lipid14: The Amber Lipid Force Field.* Journal of Chemical Theory and Computation, 2014. **10**(2): p. 865-879.

33. Tsui, V. and A. Case David, *Theory and applications of the generalized born solvation model in macromolecular simulations.* Biopolymers, 2001. **56**(4): p. 275-291.

34. Darden, T., D. York, and L. Pedersen, *Particle mesh Ewald: An N·log(N) method for Ewald sums in large systems.* The Journal of Chemical Physics, 1993. **98**(12): p. 10089-10092.

35. Deserno, M. and C. Holm, *How to mesh up Ewald sums. I. A theoretical and numerical comparison of various particle mesh routines.* The Journal of Chemical Physics, 1998. **109**(18): p. 7678-7693.

36. Deserno, M. and C. Holm, *How to mesh up Ewald sums. II. An accurate error estimate for the particle–particle–particle-mesh algorithm.* The Journal of Chemical Physics, 1998. **109**(18): p. 7694-7701.

37. Anandakrishnan, R., A. Drozdetski, Ross C. Walker, and Alexey V. Onufriev, *Speed of Conformational Change: Comparing Explicit and Implicit Solvent Molecular Dynamics Simulations.* Biophysical Journal, 2015. **108**(5): p. 1153-1164.

38. Cheatham, T.E., III, J.L. Miller, T. Fox, T.A. Darden, and P.A. Kollman, *Molecular Dynamics Simulations on Solvated Biomolecular Systems: The Particle Mesh Ewald Method Leads to Stable Trajectories of DNA, RNA, and Proteins.* Journal of the American Chemical Society, 1995. **117**(14): p. 4193-4194.

39. Toukmaji, A.Y. and J.A. Board Jr, *Ewald summation techniques in perspective: a survey.* Computer physics communications, 1996. **95**(2-3): p. 73-92.

40. Wu, X. and B.R. Brooks, *Isotropic periodic sum: A method for the calculation of long-range interactions.* The Journal of Chemical Physics, 2005. **122**(4): p. 044107.

41. Tironi, I.G., R. Sperb, P.E. Smith, and W.F. van Gunsteren, *A generalized reaction field method for molecular dynamics simulations.* The Journal of Chemical Physics, 1995. **102**(13): p. 5451-5459.

42. Ryckaert, J.-P., G. Ciccotti, and H.J.C. Berendsen, *Numerical integration of the cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes.* Journal of Computational Physics, 1977. **23**(3): p. 327-341.

43. Andersen, H.C., *Rattle: A "velocity" version of the shake algorithm for molecular dynamics calculations.* Journal of Computational Physics, 1983. **52**(1): p. 24-34.

44. Miyamoto, S. and A. Kollman Peter, *Settle: An analytical version of the SHAKE and RATTLE algorithm for rigid water models.* Journal of Computational Chemistry, 1992. **13**(8): p. 952-962.

45. Jorgensen, W.L., J. Chandrasekhar, J.D. Madura, R.W. Impey, and M.L. Klein, *Comparison of simple potential functions for simulating liquid water.* The Journal of Chemical Physics, 1983. **79**(2): p. 926-935.

46. Mark, P. and L. Nilsson, *Structure and Dynamics of the TIP3P, SPC, and SPC/E Water Models at 298 K.* The Journal of Physical Chemistry A, 2001. **105**(43): p. 9954-9960.

47. Harrison, R.W., *Stiffness and energy conservation in molecular dynamics: An improved integrator.* Journal of computational chemistry, 1993. **14**(9): p. 1112-1122.

48. Grubmüller, H., H. Heller, A. Windemuth, and K. Schulten, *Generalized Verlet Algorithm for Efficient Molecular Dynamics Simulations with Long-range Interactions.* Molecular Simulation, 1991. **6**(1-3): p. 121-142.

49. Buneman, O., *Time-reversible difference procedures.* Journal of Computational Physics, 1967. **1**(4): p. 517-535.

50. Andersen, H.C., *Molecular dynamics simulations at constant pressure and/or temperature.* The Journal of Chemical Physics, 1980. **72**(4): p. 2384-2393.

51. Lemons, D.S. and A. Gythiel, *Paul Langevin's 1908 paper "On the Theory of Brownian Motion" ["Sur la théorie du mouvement brownien," C. R. Acad. Sci. (Paris) 146, 530–533 (1908)]*. American Journal of Physics, 1997. **65**(11): p. 1079-1081.

52. Berendsen, H.J.C., J.P.M. Postma, W.F. van Gunsteren, A. DiNola, and J.R. Haak, *Molecular dynamics with coupling to an external bath.* The Journal of Chemical Physics, 1984. **81**(8): p. 3684-3690.

53. Evans, D.J. and B.L. Holian, *The Nose–Hoover thermostat.* The Journal of Chemical Physics, 1985. **83**(8): p. 4069-4074.

54. Faller, R. and J.J. de Pablo, *Constant pressure hybrid Molecular Dynamics–Monte Carlo simulations.* The Journal of Chemical Physics, 2001. **116**(1): p. 55-59.

55. Marsaglia, G. and A. Zaman, *A New Class of Random Number Generators.* Ann. Appl. Probab., 1991. **1**(3): p. 462-480.

56. Rahman, A., *Correlations in the Motion of Atoms in Liquid Argon.* Physical Review, 1964. **136**(2A): p. A405-A411.

57. Karplus, M. and J.A. McCammon, *Molecular dynamics simulations of biomolecules.* Nature Structural Biology, 2002. **9**: p. 646.

58. Zwier, M.C. and L.T. Chong, *Reaching biological timescales with all-atom molecular dynamics simulations.* Current Opinion in Pharmacology, 2010. **10**(6): p. 745-752.

59. Dong, X., J. Zhao, and Y. Xie, *Fabrication Cost Analysis and Cost-Aware Design Space Exploration for 3-D ICs.* IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2010. **29**(12): p. 1959-1972.

60. Research, T., *AMD Epyc Empowers Single-Socket Servers.* 2017.

61. Dolbeau, R., *Theoretical peak FLOPS per instruction set: a tutorial.* The Journal of Supercomputing, 2018. **74**(3): p. 1341-1377.

62. Thornton, J.E., *The CDC 6600 Project.* Annals of the History of Computing, 1980. **2**(4): p. 338-348.

63. David E. Shaw, M.M.D., Ron O. Dror, Jeffrey S. Kuskin, Richard H. Larson, John K. Salmon, Cliff Young, Brannon Batson, Kevin J. Bowers, Jack C. Chao, Michael P. Eastwood, Joseph Gagliardo, J. P. Grossman, C. Richard Ho, Douglas J. Ierardi, István Kolossváry, John L. Klepeis, Timothy Layman, Christine McLeavey, Mark A. Moraes, Rolf Mueller, Edward C. Priest, Yibing Shan, Jochen Spengler, Michael Theobald, Brian Towles, and Stanley C. Wang, *Anton, a special-purpose machine for molecular dynamics simulation.* Commun. ACM, 2008. **51**(7): p. 8.

64. Shaw, D.E., J.P. Grossman, J.A. Bank, B. Batson, J.A. Butts, J.C. Chao, M.M. Deneroff, R.O. Dror, A. Even, C.H. Fenton, A. Forte, J. Gagliardo, G. Gill, B. Greskamp, C.R. Ho, D.J. Ierardi, L. Iserovich, J.S. Kuskin, R.H. Larson, T. Layman, L.S. Lee, A.K. Lerer, C. Li, D. Killebrew, K.M. Mackenzie, S.Y.H. Mok, M.A. Moraes, R. Mueller, L.J. Nociolo, J.L. Peticolas, T. Quan, D. Ramot, J.K. Salmon, D.P. Scarpazza, U.B. Schafer, N. Siddique, C.W. Snyder, J. Spengler, P.T.P. Tang, M. Theobald, H. Toma, B. Towles, B. Vitale, S.C. Wang, and C. Young. *Anton 2: Raising the Bar for Performance and Programmability in a Special-Purpose Molecular Dynamics Supercomputer*. in *SC14: International Conference for High Performance Computing, Networking, Storage and Analysis*. 2014.

65. Höfinger, S., *Solving the Poisson–Boltzmann equation with the specialized computer chip MD-GRAPE-2.* Journal of Computational Chemistry, 2005. **26**(11): p. 1148-1154.

66. Kikugawa, G., R. Apostolov, N. Kamiya, M. Taiji, R. Himeno, H. Nakamura, and Y. Yonezawa, *Application of MDGRAPE-3, a special purpose board for molecular dynamics simulations, to periodic biomolecular systems.* Journal of Computational Chemistry, 2008. **30**(1): p. 110-118.

67. Ohmura, I., G. Morimoto, Y. Ohno, A. Hasegawa, and M. Taiji, *MDGRAPE-4: a special-purpose computer system for molecular dynamics simulations.* Philosophical transactions. Series A, Mathematical, physical, and engineering sciences, 2014. **372**(2021): p. 20130387.

68. Le Grand, S., A.W. Götz, and R.C. Walker, *SPFP: Speed without compromise—A mixed precision model for GPU accelerated molecular dynamics simulations.* Computer Physics Communications, 2013. **184**(2): p. 374-380.

69. Thall, A., *Extended-precision floating-point numbers for GPU computation*, in *ACM SIGGRAPH 2006 Research posters*. 2006, ACM: Boston, Massachusetts. p. 52.

70. Wijman, T. *Mobile Revenues Account for More Than 50% of the Global Games Market as It Reaches $137.9 Billion in 2018*. 2018 [cited 2018 8/2/2018]; Available from: https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/.

71. Foundation, N.S. *FY 2018 Budget Request to Congress*. 2016 8/2/2018]; Available from: https://www.nsf.gov/about/budget/fy2018/pdf/03_fy2018.pdf.

72. Jones, G., P. Willett, R.C. Glen, A.R. Leach, and R. Taylor, *Development and validation of a genetic algorithm for flexible docking11Edited by F. E. Cohen.* Journal of Molecular Biology, 1997. **267**(3): p. 727-748.

73. Friesner, R.A., R.B. Murphy, M.P. Repasky, L.L. Frye, J.R. Greenwood, T.A. Halgren, P.C. Sanschagrin, and D.T. Mainz, *Extra Precision Glide: Docking and Scoring Incorporating a Model of Hydrophobic Enclosure for Protein−Ligand Complexes.* Journal of Medicinal Chemistry, 2006. **49**(21): p. 6177-6196.

74. Trott, O. and A.J. Olson, *AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization and multithreading.* Journal of computational chemistry, 2010. **31**(2): p. 455-461.

75. Bryngelson Joseph, D., N. Onuchic José, D. Socci Nicholas, and G. Wolynes Peter, *Funnels, pathways, and the energy landscape of protein folding: A synthesis.* Proteins: Structure, Function, and Bioinformatics, 1995. **21**(3): p. 167-195.

76. Sagnella, D.E., J.E. Straub, and D. Thirumalai, *Time scales and pathways for kinetic energy relaxation in solvated proteins: Application to carbonmonoxy myoglobin.* The Journal of Chemical Physics, 2000. **113**(17): p. 7702-7711.

77. Chung, H.S., S. Piana-Agostinetti, D.E. Shaw, and W.A. Eaton, *Structural origin of slow diffusion in protein folding.* Science, 2015. **349**(6255): p. 1504.

78. Avinash Sodani, R.G., Jesus Corbal, Ho-Seop Kim, Krishna Vinod, Sundaram Chinthamani, Steven Hutsell, Rajat Agarwal, Yen-Chen Liu *KNIGHTS LANDING: SECONDGENERATION INTEL XEON PHI PRODUCT.* IEE, 2016(March/April 2016): p. 13.

79. van Maaren, P.J. and D. van der Spoel, *Molecular Dynamics Simulations of Water with Novel Shell-Model Potentials.* The Journal of Physical Chemistry B, 2001. **105**(13): p. 2618-2626.

80. Bowers, K.J., R.O. Dror, and D.E. Shaw, *The midpoint method for parallelization of particle simulations.* The Journal of Chemical Physics, 2006. **124**(18): p. 184109.

81. Jung, J., T. Mori, C. Kobayashi, Y. Matsunaga, T. Yoda, M. Feig, and Y. Sugita, *GENESIS: a hybrid-parallel and multi-scale molecular dynamics simulator with enhanced sampling algorithms for biomolecular and cellular simulations.* Wiley Interdisciplinary Reviews. Computational Molecular Science, 2015. **5**(4): p. 310-323.

82. Jung, J., C. Kobayashi, T. Imamura, and Y. Sugita, *Parallel implementation of 3D FFT with volumetric decomposition schemes for efficient molecular dynamics simulations.* Computer Physics Communications, 2016. **200**: p. 57-65.

83. Jung, J., T. Mori, and Y. Sugita, *Midpoint cell method for hybrid (MPI+OpenMP) parallelization of molecular dynamics simulations.* Journal of Computational Chemistry, 2014. **35**(14): p. 1064-1072.

84. Fitch, B.G., A. Rayshubskiy, M. Eleftheriou, T.J.C. Ward, M.E. Giampapa, M.C. Pitman, J.W. Pitera, W.C. Swope, and R.S. Germain, *Blue Matter: Scaling of N-body simulations to one atom per node.* IBM Journal of Research and Development, 2008. **52**(1.2): p. 145-158s.

85. Kollman, P.A., I. Massova, C. Reyes, B. Kuhn, S. Huo, L. Chong, M. Lee, T. Lee, Y. Duan, W. Wang, O. Donini, P. Cieplak, J. Srinivasan, D.A. Case, and T.E. Cheatham, *Calculating Structures and Free Energies of Complex Molecules: Combining Molecular Mechanics and Continuum Models.* Accounts of Chemical Research, 2000. **33**(12): p. 889-897.

86. Abel, R., L. Wang, E.D. Harder, B.J. Berne, and R.A. Friesner, *Advancing Drug Discovery through Enhanced Free Energy Calculations.* Accounts of Chemical Research, 2017. **50**(7): p. 1625-1632.

87. Kuhn, B., M. Tichý, L. Wang, S. Robinson, R.E. Martin, A. Kuglstatter, J. Benz, M. Giroud, T. Schirmeister, R. Abel, F. Diederich, and J. Hert, *Prospective Evaluation of Free Energy Calculations for the Prioritization of Cathepsin L Inhibitors.* Journal of Medicinal Chemistry, 2017. **60**(6): p. 2485-2497.

88. Yu, H.S., Y. Deng, Y. Wu, D. Sindhikara, A.R. Rask, T. Kimura, R. Abel, and L. Wang, *Accurate and Reliable Prediction of the Binding Affinities of Macrocycles to Their Protein Targets.* Journal of Chemical Theory and Computation, 2017. **13**(12): p. 6290-6300.

89. Wang, L., Y. Deng, Y. Wu, B. Kim, D.N. LeBard, D. Wandschneider, M. Beachy, R.A. Friesner, and R. Abel, *Accurate Modeling of Scaffold Hopping Transformations in Drug Discovery.* Journal of Chemical Theory and Computation, 2017. **13**(1): p. 42-54.

90. Harder, E., W. Damm, J. Maple, C. Wu, M. Reboul, J.Y. Xiang, L. Wang, D. Lupyan, M.K. Dahlgren, J.L. Knight, J.W. Kaus, D.S. Cerutti, G. Krilov, W.L. Jorgensen, R. Abel, and R.A. Friesner, *OPLS3: A Force Field Providing Broad Coverage of Drug-like Small Molecules and Proteins.* Journal of Chemical Theory and Computation, 2016. **12**(1): p. 281-296.

91. Wang, L., Y. Wu, Y. Deng, B. Kim, L. Pierce, G. Krilov, D. Lupyan, S. Robinson, M.K. Dahlgren, J. Greenwood, D.L. Romero, C. Masse, J.L. Knight, T. Steinbrecher, T. Beuming, W. Damm, E. Harder, W. Sherman, M. Brewer, R. Wester, M. Murcko, L. Frye, R. Farid, T. Lin, D.L. Mobley, W.L. Jorgensen, B.J. Berne, R.A. Friesner, and R. Abel, *Accurate and Reliable Prediction of Relative Ligand Binding Potency in Prospective Drug Discovery by Way of a Modern Free-Energy Calculation Protocol and Force Field.* Journal of the American Chemical Society, 2015. **137**(7): p. 2695-2703.

92. Lee, T.-S., Y. Hu, B. Sherborne, Z. Guo, and D.M. York, *Toward Fast and Accurate Binding Affinity Prediction with pmemdGTI: An Efficient Implementation of GPU-Accelerated Thermodynamic Integration.* Journal of Chemical Theory and Computation, 2017. **13**(7): p. 3077-3084.

93. Mermelstein Daniel, J., C. Lin, G. Nelson, R. Kretsch, J.A. McCammon, and C. Walker Ross, *Fast and flexible gpu accelerated binding free energy calculations within the amber molecular dynamics package.* Journal of Computational Chemistry, 2018. **0**(0).

94. Beutler, T.C., A.E. Mark, R.C. van Schaik, P.R. Gerber, and W.F. van Gunsteren, *Avoiding singularities and numerical instabilities in free energy calculations based on molecular simulations.* Chemical Physics Letters, 1994. **222**(6): p. 529-539.

95. Luzhkov, V.B., *On relation between the free-energy perturbation and Bennett's acceptance ratio methods: Tracing the influence of the energy gap.* The Journal of Chemical Physics, 2010. **132**(19): p. 194104.

96. Shirts, M.R. and J.D. Chodera, *Statistically optimal analysis of samples from multiple equilibrium states.* The Journal of Chemical Physics, 2008. **129**(12): p. 124105.

97. Kumar, A., B. Rani, R. Sharma, G. Kaur, R. Prasad, A. Bahl, and M. Khullar, *ACE2, CALM3 and TNNI3K polymorphisms as potential disease modifiers in hypertrophic and dilated cardiomyopathies*. Vol. 438. 2017. 1-8.

98. Vagnozzi, R.J., G.J. Gatto, L.S. Kallander, N.E. Hoffman, K. Mallilankaraman, V.L.T. Ballard, B.G. Lawhorn, P. Stoy, J. Philp, A.P. Graves, Y. Naito, J.J. Lepore, E. Gao, M. Madesh, and T. Force, *Inhibition of the Cardiomyocyte-Specific Kinase TNNI3K Limits Oxidative Stress, Injury, and Adverse Remodeling in the Ischemic Heart.* Science translational medicine, 2013. **5**(207): p. 207ra141-207ra141.

99. Lawhorn, B.G., J. Philp, A.P. Graves, D.A. Holt, G.J. Gatto, and L.S. Kallander, *Substituent Effects on Drug–Receptor H-bond Interactions: Correlations Useful for the Design of Kinase Inhibitors.* Journal of Medicinal Chemistry, 2016. **59**(23): p. 10629-10641.

100. Lingle Wang, T.L., Robert Abel *Cycle closure estimation of relative binding affinities and errors*. 2013, Schrodinger, Llc.

101. Gathiaka, S., S. Liu, M. Chiu, H. Yang, J.A. Stuckey, Y.N. Kang, J. Delproposto, G. Kubish, J.B. Dunbar, H.A. Carlson, S.K. Burley, W.P. Walters, R.E. Amaro, V.A. Feher, and M.K. Gilson, *D3R Grand Challenge 2015: Evaluation of Protein-Ligand Pose and Affinity Predictions.* Journal of computer-aided molecular design, 2016. **30**(9): p. 651-668.

102. Boresch, S., F. Tettinger, M. Leitgeb, and M. Karplus, *Absolute Binding Free Energies: A Quantitative Approach for Their Calculation.* The Journal of Physical Chemistry B, 2003. **107**(35): p. 9535-9551.

103. Kim, M.O., P.G. Blachly, J.W. Kaus, and J.A. McCammon, *Protocols Utilizing Constant pH Molecular Dynamics to Compute pH-Dependent Binding Free Energies.* The Journal of Physical Chemistry B, 2015. **119**(3): p. 861-872.

104. Christ Clara, D., E. Mark Alan, and F. van Gunsteren Wilfred, *Basic ingredients of free energy calculations: A review.* Journal of Computational Chemistry, 2009. **31**(8): p. 1569-1582.

105. Pohorille, A., C. Jarzynski, and C. Chipot, *Good Practices in Free-Energy Calculations.* The Journal of Physical Chemistry B, 2010. **114**(32): p. 10235-10253.

106. Hamelberg, D., J. Mongan, and J.A. McCammon, *Accelerated molecular dynamics: A promising and efficient simulation method for biomolecules.* The Journal of Chemical Physics, 2004. **120**(24): p. 11919-11929.

107. Miao, Y., V.A. Feher, and J.A. McCammon, *Gaussian Accelerated Molecular Dynamics: Unconstrained Enhanced Sampling and Free Energy Calculation.* Journal of Chemical Theory and Computation, 2015. **11**(8): p. 3584-3595.

108. Sugita, Y. and Y. Okamoto, *Replica-exchange molecular dynamics method for protein folding.* Chemical Physics Letters, 1999. **314**(1): p. 141-151.

109. Wong, K.-Y. and B.M. Pettitt, *A new boundary condition for computer simulations of interfacial systems.* Chemical Physics Letters, 2000. **326**(3): p. 193-198.

110. Skjevik, A.A., B.D. Madej, C.J. Dickson, C. Lin, K. Teigen, R.C. Walker, and I.R. Gould, *Simulation of lipid bilayer self-assembly using all-atom lipid force fields.* Physical Chemistry Chemical Physics, 2016. **18**(15): p. 10573-10584.

111. Klauda, J.B., R.M. Venable, J.A. Freites, J.W. O'Connor, D.J. Tobias, C. Mondragon-Ramirez, I. Vorobyov, A.D. MacKerell, and R.W. Pastor, *Update of the CHARMM All-Atom Additive Force Field for Lipids: Validation on Six Lipid Types.* The Journal of Physical Chemistry B, 2010. **114**(23): p. 7830-7843.

112. Venable, R.M., Y. Luo, K. Gawrisch, B. Roux, and R.W. Pastor, *Simulations of Anionic Lipid Membranes: Development of Interaction-Specific Ion Parameters and Validation Using NMR Data.* The Journal of Physical Chemistry B, 2013. **117**(35): p. 10183-10192.

113. Venable, R.M., F.L.H. Brown, and R.W. Pastor, *Mechanical properties of lipid bilayers from molecular dynamics simulation.* Chemistry and Physics of Lipids, 2015. **192**: p. 60-74.

114. Jämbeck, J.P.M. and A.P. Lyubartsev, *Derivation and Systematic Validation of a Refined All-Atom Force Field for Phosphatidylcholine Lipids.* The Journal of Physical Chemistry B, 2012. **116**(10): p. 3164-3179.

115. Jämbeck, J.P.M. and A.P. Lyubartsev, *An Extension and Further Validation of an All-Atomistic Force Field for Biological Membranes.* Journal of Chemical Theory and Computation, 2012. **8**(8): p. 2938-2948.

116. Steinbach Peter, J. and R. Brooks Bernard, *New spherical-cutoff methods for long-range forces in macromolecular simulation.* Journal of Computational Chemistry, 1994. **15**(7): p. 667-683.

117. Dickson, C. *AMBER-Umbrella_COM_restraint_tutorial*. 2016.

118. Bemporad, D., J.W. Essex, and C. Luttmann, *Permeation of Small Molecules through a Lipid Bilayer: A Computer Simulation Study.* The Journal of Physical Chemistry B, 2004. **108**(15): p. 4875-4884.

119. Carpenter, Timothy S., Daniel A. Kirshner, Edmond Y. Lau, Sergio E. Wong, Jerome P. Nilmeier, and Felice C. Lightstone, *A Method to Predict Blood-Brain Barrier Permeability of Drug-Like Compounds Using Molecular Dynamics Simulations.* Biophysical Journal, 2014. **107**(3): p. 630-641.