2014

Coarse Grained Monte Carlo Simulation of the Self-Assembly of the HIV-1 Capsid Protein

Jeffrey Weber
University of Central Florida

COARSE GRAINED MONTE CARLO SIMULATION OF THE SELF-ASSEMBLY OF THE HIV-1 CAPSID PROTEIN

by

JEFFREY WEBER

A thesis submitted in partial fulfilment of the requirements
for the Honors in the Major Program in Physics
in the College of Sciences
and the Burnett Honors College
at the University of Central Florida
Orlando, Florida

Spring Term
2014

Thesis Chair: Bo Chen, Ph. D.

ABSTRACT

In this study, a Monte Carlo simulation was designed to observe the self-assembly of the HIV-1 capsid protein. The simulation allowed a coarse grained model of the capsid protein with defined interaction sites to move freely in three dimensions using the Metropolis criterion. Observations were made as to which parameters affected the assembly the process. The ways in which the assembly were affected were also noted. It was found that proper dimerization of the capsid protein was necessary in order for the lattice to form properly. It was also found that a strong trimeric interface could be responsible for double-layered assemblies. Further studies may be conducted by further varying of parameters or reworking the dynamics of the simulation. The possible causes of curvature within the assembly still need to be researched further.

Dedicated to my Grandmother who supported me throughout my college career.

ACKNOWLEDGMENTS

I would like to thank Bo Chen for his contributions that made this thesis possible as well as my parents for supporting me throughout the process.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

INTRODUCTION

The human immunodeficiency virus (HIV) epidemic is estimated to affect about 34.2 million people worldwide[1]. In the United States alone, it was estimated that there were 1,148,200 people living with HIV at the end of 2009, with approximately 50,000 people being infected every year.[1, 2]. Worldwide, there were 2.5 million new cases of HIV reported in 2011[1]. Since the HIV epidemic began, about 30 million people around the world have died, with an additional 1.8 million deaths every year[1].

HIV is a virus and by nature, during its lifetime will infect a host cell, copy its genetic material, and leave the cell in order to find a new host[3]. HIV belongs to a class of viruses known as retroviruses, meaning that it carries its genetic material in the form of RNA and uses the process of reverse transcription to replicate[4]. HIV can be further classified as a lentivirus due to the duration of its incubation period. Over long periods, HIV can compromise the human immune system, which allows opportunistic infections to flourish.

Despite the efforts of researchers, there is still no cure for HIV. That being said, HIV is much more treatable now than ever. One possible target for an effective HIV drug could be the viral capsid. The capsid is the part of the virus, typically made up of repeated subunits, that encloses the genetic material[3]. Some studies show that targeting the capsid can affect the infectivity of the virus[5, 6]. The infectivity of HIV-1 is dependent on the proper disassembly of the capsid, or "uncoating," in the host cell[7]. If the structure of the capsid becomes too stable, the genetic material may not be able to escape, resulting in a loss of infectivity[8, 9]. Conversely, if the capsid becomes too destabilized, the virus may not be able to form correctly[7, 8]. Shi, et al. reasoned in the study of how PF74, a small molecule HIV-1 inhibitor, affects the stability of the capsid and the infectivity of the virus, that the loss of infectivity in destabilized capsids may be due to a flaw in reverse transcription in the host cells that stems from the premature uncoating[7].

Purpose

By studying the HIV-1 capsid, it may be possible to discover new treatments or even cures for those infected with HIV. Although the structure of the subunits involved in the capsid assembly are known, relatively little is known about the assembly process of the HIV-1 capsid. Thus, the goal of this project is to learn as much about the self-assembly of the HIV-1 capsid as possible through Monte Carlo simulations. It has been shown that stabilizing and destabilizing mutations can compromise infectivity, and knowing how the capsid assembles could allow targets for potential antivirals to be determined[10].

Further, a better understanding of the HIV-1 capsid could serve purposes beyond the scope of treating those infected with the virus. It has been shown that viral capsids can be repurposed, including the possibility of targeting other infections[3]. This could make the viral capsid an effective delivery system for drugs targeting that infection. In addition to this, Soto and Ratna note that viral capsids can be useful in bionanotechnology due to the scale and adaptability of viral capsids[11].

Definition of Terms

Terms will be used related to biological elements as well as coding throughout this thesis and will be defined here. The viral capsid is a shell that encloses the genetic material within a virus. The capsid protein may be abbreviated CA. This viral capsid is made up of repeating oligomers or subunits that collect to form the shell. Oligomers are defined as collections of subunits. In the case of HIV-1 the most important oligomers are the dimer, trimer, pentamer and hexamer. These contain two, three, five and six subunit monomers, respectively. The subunit monomer is further divided into two domains, the amino-terminal domain, defined by its termination by an amino acid containing a free amine group, and the carboxyl-terminal domain, which is terminated by a carboxyl group. For simplicity, the amino-terminal domain can be simplified to NTD and

the carboxyl-terminal domain can be represented by CTD. Two of the most common structures in these domains are α helices and β sheets. The α helix is a spiral structure of connected residues and the β sheets are less common and consist of β strands. β strands are just chains of residues where the backbone atoms are mostly stretched.

Some coding terms that may be used include Monte Carlo, which is an algorithm that requires random sampling to accomplish some task. Monte carlo may be abbreviated MC, and in this simulation the Metropolis criterion is used. The Metropolis criterion uses the Boltzman factor to determine the probability that the system can make a particular move by comparing the previous energy to the energy after an MC step[12]. OpenMP is a software package that allows the parallelization of Fortran code, and may be represented by OMP. These and other significant terms may be used and defined throughout this document.

LITERATURE REVIEW

Many scientists have performed a significant amount of research on HIV, the capsid protein and the assembly process. A collection of these studies was analyzed in order to better understand the maturation of HIV and the assembly and structure of the capsid.

In HIV, Gag molecules collect at the plasma membrane of the host cell and creates a spherical structure[13, 14]. This immature shell matures while budding from the plasma membrane of the host cell, where the viral protease cleaves Gag into component proteins[14, 15]. These proteins include the matrix protein (MA), the capsid protein (CA), and the nuclear capsid protein (NC). About 1,500 of these capsid proteins would assemble to form the capsid shell around the nuclear capsid proteins[16]. Although polymorphisms exist, such as tubular assemblies, HIV-1 cores typically form into the shape of a fullerene cone in the wild type virus[14, 16, 17, 18]. The correct assembly of the HIV-1 capsid plays a vital role in determining the infectivity of the virus[19].

Structure of HIV-1 Capsid

Viral capsids are typically made up of repeating subunits that together form a fairly symmetric assembly[3]. In the case of the HIV-1 virus, these subunits are the hexamer and pentamer. The hexamer is made up of six dimer subunits and has been identified through experimentation, while the pentamer is made up of five and is thought necessary to close the conical structure, but not observed. While the pentamer subunit is needed to close the assembled capsid, pentamers have only been constructed using mutant CA proteins[18, 20]. These oligomers are arranged with the NTD on the inside, with the CTD facing outwards. Adjacent hexamers and pentamers are joined through the CTD dimerization interface as well as a CTD three-fold trimerization interface, which can be seen in Figure 2[17]. It is also thought that dimers may adjust to cover variable distances depending on whether they are connecting hexamers to hexamers and hexamers to pentamers[16].

As stated earlier, the most common HIV-1 capsid shape is considered the fullerene cone when assembled *in vivo*[18]. This cone is comprised of a lattice of about 250 hexamers and precisely 12 pentamers[7, 16]. Pentamers allow for the sharp angles needed for the lattice to form into a cone[16, 17]. The pentamers are distributed near the ends of the cone, with 5 pentamers at the sharper end and 7 at the wider end[16]. The curvature of the cone can be caused by additional factors besides these 12 pentamers. One factor contributing to the curvature is the different orientations in the CTD allowed by the flexibility of helix 9[16, 17, 19]. Variations in the NTD-CTD interfaces are also know to contribute to the curvature[19, 21]. The angle of the cone is quantized, and the wild type virus has been known to adopt one of five cone angles[18]. The most common angle is also the narrowest angle, which is much more prominent than other cone angles[13, 18].

Capsid assembly *in vitro* shows preferred shapes, but with severe polymorphism[13]. HIV-1 has been shown to self-assemble into tubes and cones *in vitro*[17]. This implies that the required information for capsid formation is contained within the capsid protein[18].

Structure of Capsid Protein

The most basic subunit of the HIV-1 capsid is the CA monomer. Each HIV-1 CA monomer contains an amino-terminal domain (NTD) and a carboxyl-terminal domain (CTD) [22, 23, 24]. The N-terminal domain consists of seven α helices and a β hairpin[22, 23]. A flexible linker region of residues and a 3_{10} helix connects the N-terminal domain to the C-terminal domain, which consists of four additional α helices[24]. This linker region is important for the correct assembly of the HIV-1 capsid[8]. The monomer subunit can be seen in Figure 1.

In solution, two CA monomers spontaneously form into a dimer at the CTD dimerization interface around helix 9 with a dissociation constant of 18 μM[24, 25, 26]. Dimerization of subunits supports capsid assembly *in vitro*[15]. The angle of the dimerization helices is confined and the dimerization interface is much stronger than other interfaces[17].

Assembly *in-vitro*

Assembly of the capsid shell can be contributed to hydrophobic, electrostatic, van der Waals, and hydrogen bonding interactions[27]. All the required information for the subunits to form into the conical capsid is contained within the CA polypeptide[18]. In the case of the HIV-1 capsid protein, interaction sites include a CTD-CTD dimerization interface at helix 9, a three-fold

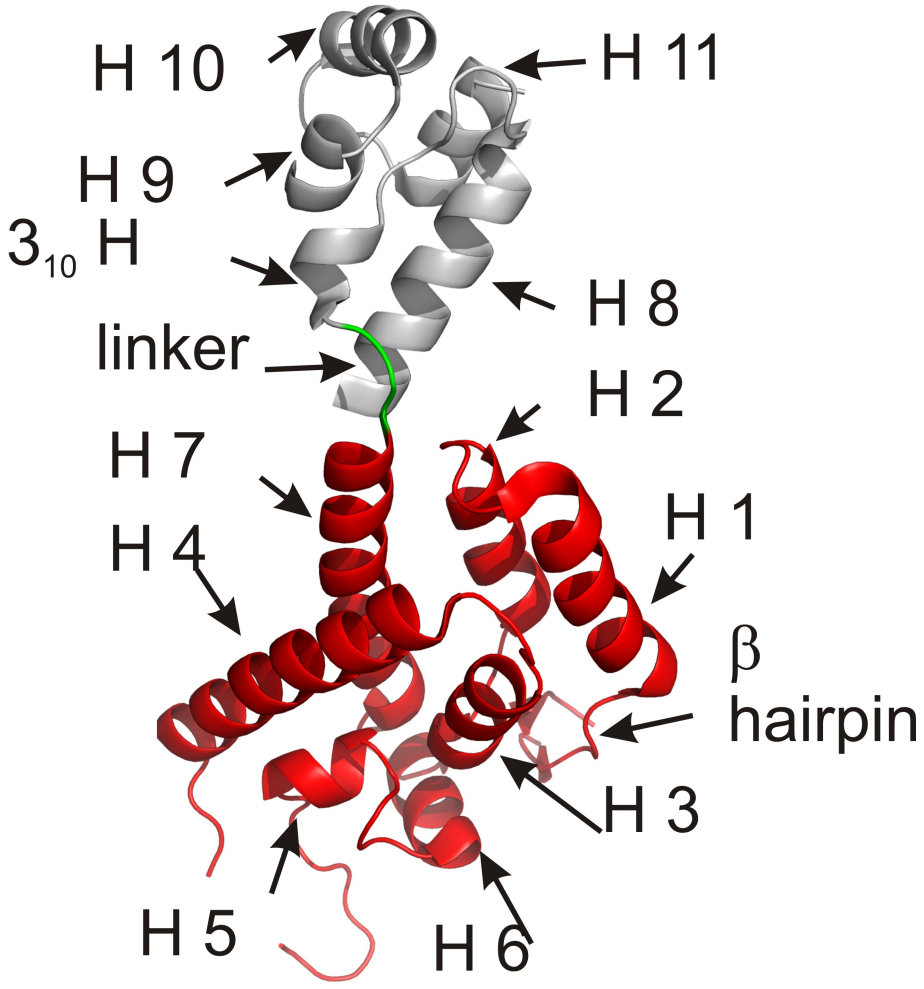

Figure 1: The structure of HIV-1 CA as derived from 3H47.pdb.

CTD-CTD trimerization interface at helix 10, an NTD-CTD intersubunit interface between helices 4 and 8, and an NTD-NTD interface between helices 2 and 3[10, 17, 24, 28]. These interaction sites can be seen in Figure 2.

Assembled structures consist of tubes and cones that resemble actual capsids, but with great uncertainty in diameters[17]. Dimerization of monomers favors the assembly of the capsid *in vitro*, even though dimers may play a different role while assembling *in vivo*[15]. In order for *in vitro* assembly to occur, the capsid protein must be in a solution with a high concentration of salt[17].

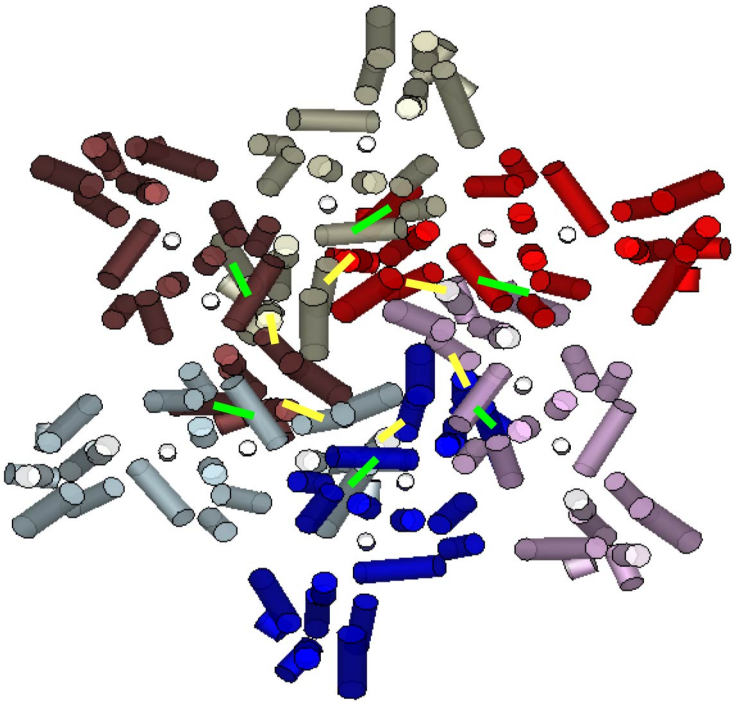

Figure 2: Lines between cylinders represent interacting interfaces. Figure adapted from Chen and Tycko[28].

Simulations

Due to the speed of assembly, it can be difficult to find assembly intermediates and characterize the assembly of viral capsids in detail through experimentation alone[27, 28]. Through the use of simulations based on experimental data, further investigation into assembly pathways can be performed[27, 28]. While a full atomic representation of the entire capsid assembly is not currently practical over long time scales, coarse grained models can be used to determine assembly pathways[27, 28].

One benefit of coarse grain models is that you can adjust each factor separately in order to see how it affects the assembly of the capsid[27]. This can give insights into how different interfaces can affect the assembly process. Keeping interaction strengths weak can aid in the assembly process.[27]. If the interaction strength is too high, a "kinetic trap" is created, which may stop subunits from being able to form the full capsid[15]. While interaction strengths should be kept weak, they cannot be too weak or no assembly will be observed[15]. Optimal assembly should occur when the subunit associations can be reversed[27].

RESEARCH QUESTION

Methods

The self assembly of the HIV-1 capsid protein will be analyzed through an MC simulation written in Fortran 90. Code can be produced to use either the monomer or the dimer as the most basic subunit of assembly. Monte Carlo simulations consist of using random numbers to solve a problem, in this case the Metropolis criterion will be used to determine the movement of molecules in the system. The simulation maintains a periodic boundary condition while allowing molecules to explore 3D configurations.

The simulation is broken down into several main steps. The system initially sets up an evenly spaced lattice of subunits and allows them to anneal at a high temperature for 10,000 steps per subunit. The annealing process is designed to randomize the system in order to replicate actual systems. Alternatively, a previously defined lattice can be read into the program, and the simulation can be started in this predefined configuration. When one of these processes has been completed, subunit assembly can begin. Assembly consists of looping through each molecule in the system and attempting to randomly translate and rotate the molecule. Maximum steps of 0.05 nm in each Cartesian direction and maximum rotations of $\pi/360$ radians can be defined. If the move satisfies all criteria specified in the next paragraph, then the move is accepted and the position and orientation of the molecule is updated. If the move is rejected, the molecule's position and orientation will not be updated. The simulation will then select the next molecule and continue until the simulation is stopped.

Each step can be broken down into several parts. After the molecule is selected, random numbers are used to determine how the molecule should be translated and rotated. It should be noted that the translation must be calculated in all Cartesian directions.In its new position, the subunit must meet several criteria in order to make sure that the move is accepted. First, it must not

collide with any other molecules, so an algorithm will be needed to check for collisions. Additionally, the energy of the system must decrease or pass a random check that is further explained in the Energy Calculation section. If either of these criteria are not met, then the move will be rejected and the next molecule will be selected. If both criteria are satisfied, the move will be accepted, the appropriate values will be updated, and the next molecule will be selected.

Every time the simulation reaches a certain number of steps, the output will be written to file. To save space, only three points from each subunit need to be recorded. From this output, the system can be reconstructed using visualization software. Conclusions about how the system formed can then be derived from images and movies created using this visualization software.

Translation and Rotation

Once a molecule has been selected, it must be moved to a new position in order for the system to evolve. Translation could occur in all Cartesian directions and can be determined by producing a random array of three real numbers within an allowed range. This process is defined by the equation

$$trsl = rand \times 2.0 \times trsl_{max} - trsl_{max} \tag{1}$$

where rand is a randomly generated number between zero and one. The maximum translation in any cartesian direction was taken to be 0.05 nm. Similarly, an axis of rotation for the molecule can be chosen and a rotation angle be defined the equation

$$angle = rand \times 2.0 \times angle_{max} - angle_{max} \tag{2}$$

where rand is a random number between zero and one. The maximum change of angle was taken to be $\pi/360$ radians. This angle than then be used to derive a rotation matrix

$$\begin{pmatrix} u_x^2 + (1-u_x^2)cs & u_x \times u_y(1-cs) + u_z \times sn & u_x \times u_z(1-cs) - u_y \times sn \\ u_x \times u_y(1-cs) - u_z \times sn & u_y^2 + (1-u_y^2)cs & u_y \times u_z(1-cs) + u_x \times sn \\ u_x \times u_z(1-cs) + u_y \times sn & u_y \times u_z(1-cs) - u_x \times sn & u_z^2 + (1-u_z^2)cs \end{pmatrix} \quad (3)$$

where u is the position vector, sn is $sin(\theta)$, and cs is $cos(\theta)$. This matrix can be applied to rotate the molecule. Assuming that the energy and collision criteria are met, these new coordinates will replace the old coordinates and represent the new location of the molecule.

Collision Detection

In order for the system to be considered valid, molecules must not collide with one another. As such, a collision detection algorithm must be implemented each time a molecule is moved. The collision detection algorithm consists of three main checks, with each subsequent check on a smaller scale than the last. The reason why three levels of checks were implemented is to avoid unnecessary computation. If any level determines that the subunits cannot collide, then subsequent calculations are not necessary, and the simulation can move on to the next subunit in the system. Collision detection must be run over every molecule in the system, or every molecule in the verlet list if implemented.

The first check assumes each molecule is a sphere and calculates the distance between the center of each molecule. The radius of the sphere is assumed to be equal to the maximum reach of the molecule. The distance between the center of the two molecules is then calculated. If it is found that these spheres do not collide, then there is no possibility of collision. If the spheres do collide, then the next level of check must be performed.

The second check must loop over every helix in each of the molecules. The idea again is

to find the distances between the center of the two helices and see if the it is greater than the sum of their radii. If the distance is greater that the reach of the helices, then they do not collide and the next set of helices can be considered. If the reach is greater than the distance, then they may or may not collide and an additional check is needed.

The last level of the collision detection algorithm actually treats two helices as cylinders. If the closest point on their axes is less than the sum of their radii, then the cylinders are said to collide. This is accomplished through an algorithm designed by Dr. Chen[29]. If there is no collision between the cylinders, then the collision detection algorithm is allowed to continue. If a collision between two cylinder is detected, then the algorithm can return immediately that a collision has been found.

Energy Calculation

In order to determine the likelihood of a certain configuration, the energy of each configuration must be evaluated. Statistical physics tells us that the system should slowly move towards the state with the least amount of energy associated with it. Because only one molecule is moved at a time, the change in local energy of the displaced molecule is all that needs to be calculated. The local energy between the selected molecule and all interacting molecules must be determined.

The energy calculation algorithm is divided into two parts. Similar to the collision detection algorithm, the algorithm will treat each molecule as spheres and check to see if they are within a particular cutoff limit. The cutoff limit is defined as the distance at which intermolecular interaction strengths are too insignificant to contribute to the total energy, taken to be 10.91037 nm. If the molecules are too far apart, the algorithm can move on to the next molecule. Otherwise, the energy between interaction sites on the molecules must be determined.

The next part determines the interaction energy on a site-by-site basis. The interaction potential was determined using the Lennard-Jones potential as well as and angular restraint. The energy was obtained using

$$V(\theta, r) = f(\theta) \cdot g(r) \tag{4}$$

$$f(\theta) = \begin{cases} \frac{|\theta - \theta_0|}{\theta_{max}} - 1 & |\theta - \theta_0| \leq \theta_{max} \\ 0 & |\theta - \theta_0| > \theta_{max} \end{cases} \tag{5}$$

$$g(r) = 4\epsilon_0 \left[\left(\frac{\sigma}{r} \right)^{12} - \left(\frac{\sigma}{r} \right)^{6} \right]. \tag{6}$$

Equation (5) places an angular dependence on the interactions, where θ is the angle between interacting cylinders in the simulation, θ_0 is the angle derived from experimental data, and θ_{max} is the cutoff angle at which the interaction strength becomes zero. Equation (6) is the Lennard-Jones potential, where r is the distance between interaction cites in the simulation and σ is the optimal distance between interaction sites. Values used can be found in Table 1. This method of calculating energy has been applied to many course grained models, such as Chen and Tycko[28].

Once the local energy has been determined, it can be used to find the change in total energy. Metropolis criterion was then used to determine whether the move should be accepted or rejected, assuming it passes the collision criterion.

Monte-Carlo Considerations

Monte-Carlo methods use systematic random sampling to evolve the system into a preferred state. An effective Monte-Carlo simulation requires a reliable random number generator.

Table 1: Parameters used in the evaluation of energy for the course grained model in the simulation.

Interface	$\theta_0(radians)$	$\sigma(nm)$
NTD-NTD	0.690483	0.9702867
NTD-CTD	1.218533	0.7257217
CTD-CTD	0.5061	0.982183

In this simulation, the Fortran intrinsic function for calling random numbers can be used that is seeded using the current date and time. In this simulation, random numbers will be used at three main points in the code. The first is to determine the direction and magnitude of the translation, the second determines the magnitude of rotation, and the third is to check that the energy satisfies the Metropolis criterion.

Visualization

Although Fortran is efficient at simulating the assembly of the HIV-1 capsid, it does not provide a visual output from which conclusions can be drawn. Visualization must be implemented using software such as MATLAB or Visual Molecular Dynamics (VMD). This means that the Fortran simulation must output the state of the system to text files at reasonable intervals. Although the entire system could be written to file, it is much more space efficient if only three points from each subunit are recorded. Since this is a coarse grained model, each molecule is treated as a rigid body and therefore a template can be mapped exactly to the correct position using only these three points.

VMD provides a console from which TCL scripts can be run, making it a very versatile tool in visualizing molecular systems. TCL scripts can be devised to draw systems, highlight features, and make movies. TCL scripts can be used to transform a particular three point representation into a full cylindrical representation, and then implement VMD's draw commands to draw cylinders. For ease of viewing, the N-terminus and C-terminus can be colored differently in the VMD window.

The idea of drawing the system can be expanded into creating a movie. Because output was recorded at certain intervals, many systems can be drawn and snapshots can be produced at each step. This is best done using a TCL script that implements the drawing script discussed earlier. These snapshots can then be combined into a movie. Unfortunately, this is a static view of the system that can not be probed further once it has been created. A more dynamic way

to create a movie is to use VMD's ability to map trajectories. This means that the three point representation must be converted to cylindrical as before, but instead of drawing cylinders, VMD will draw "bonded atoms." These bonded atoms can be displayed as uncapped cylinders or using VMD's "licorice" representation. This allows full functionality of VMD at all steps, including the ability to view the system from different perspectives and focus on remarkable aspects of the assembly. This trajectory can even be made into a static mpeg movie as before using VMD's Movie Maker extension.

Optimizations

Optimizations on the code are necessary to ensure that results are obtained in the least amount of time possible. These optimizations come from the programmer as well as the compiler. It is necessary to ensure that these optimizations do not compromise the codes integrity, and that the simulation still runs as intended.

Verlet List

An optimization that can be implemented by the programmer is the inclusion of a verlet list. The verlet list is an array that stores a list of all molecules that are within a cutoff distance that can be determined from the interaction cutoff distance. If a molecule is within this range, it will be added to the verlet list. The verlet list must be updated whenever a molecule has moved far enough that it can potentially interact with molecules that were not previously in the verlet list. To maximize the efficiency of the verlet list, the verlet cutoff distance must be small enough to significantly limit the number of molecules, but large enough that the list must not constantly be reset.

By using a verlet list, the number of molecules that must be checked in the collision detection and energy calculation algorithms can be severely reduced. These methods must be modified to loop over molecules in the verlet list instead of all molecules in the system. Although the verlet

list takes some time to set up, the speed boost in these two algorithms reduces the overall runtime. This boost can be further improved by sorting the verlet list by distance from the source molecule, allowing the collision detection algorithm a better shot at detecting the collision and exiting early.

OpenMP

Multi-threading is a way to squeeze more performance out of algorithms by parallelizing them. An algorithm is a good candidate for parallelization if it is running a large number of repeated tasks. For this reason, the collision detection, energy calculation, and verlet list algorithms are good candidates for parallelization. The loop over one molecule is not a good candidate for parallelization because if two molecules are moved simultaneously, the collision detection and energy calculation algorithms may return incorrect values, which can lead to collisions of molecules or moves that are accepted or rejected based on incorrect energy values.

OpenMP provides an easy way for programmers to parallelize their Fortran codes. OpenMP uses a system of directives that can be used to create parallel regions in the code. The speed increase that can be obtained for a given region is not directly proportional to the number of threads it implements, as the set up cost must be taken into account. This means that for algorithms that do not require a large number of steps, OpenMP may actually take more time to set up the parallel regions than can be saved by the parallelization. Because of this, algorithms must be analyzed to see if there is actually any benefit to parallelization. The increase in speed is also dependent on which variables are made private and which are public to all threads, so these must be checked to see how the best possible runtime can be obtained.

The collision detection algorithm can benefit from parallelization using an `OMP PARALLEL DO` loop. The algorithm must loop over a large number of molecules, and must do a significant amount of work for each. In order to make sure each thread knows if there is a collision, an `OMP ATOMIC` call can be used to synchronize these values without delaying the algorithm. The collision detection can benefit from returning early if a collision is found, but OpenMP requires that all

loops finish before the parallel region can be ended. To accommodate for this, a flag can be used to skip the work in a loop if the collision criterion has been triggered.

The energy calculation algorithm does not contain the same runtime complexity as the collision algorithm, but can still benefit from parallelization using `OMP PARALLEL SECTIONS`. This will allow the energy of the system before the move and after the move to be calculated in parallel. Because these two separate energy values are calculated in sections, they do not need to call `OMP ATOMIC` like the collision detection algorithm needed to.

The verlet list algorithm is parallelized in a manner similar to the collision detection algorithm using `OMP PARALLEL DO`. Unlike the collision detection algorithm, the verlet list algorithm should not try to access the same variable, and will not end early, so it is a straightforward optimization.

Compilers

The compiler used can also have some effect on the speed of the simulation. Two compilers that are readily available are the Intel® Fortran compiler(ifort) and the GNU Fortran compiler(gfortran). Although the compiled codes should perform identical tasks, the compilers may optimize the code in different ways. This means that one of these compilers may produce a faster simulation than the other. Each of these compilers come with their own set of compilation flags that optimize the code in different ways. By choosing the correct compiler and optimizations, the code can be tuned to run more efficiently while maintaining accuracy.

Algorithm Organization

While it may seem trivial, the organization of the collision detection and energy calculation algorithms is vital in determining the speed of the simulation. If the faster of these two algorithms has to wait on the slower one, then the code is not running as efficiently as it could. The collision detection algorithm requires significantly more computational time than the energy calculation

algorithm, so the energy criterion should be checked first. As long as all criteria are satisfied, it does not matter in which order they are checked. If the energy criterion is not satisfied, the code can skip the costly collision detection algorithm, providing the simulation with a boost in speed. The breakdown of how much time the simulation devoted to each algorithm during a short simulation can be found in Figure 3.

Model

The HIV-1 capsid protein consists of 11 α helices and a 3_{10} helix. In this course grained model, each of these helices will be represented by a cylinder that represents the helix. In the model, the 3_{10} helix is labeled as helix 8. Due to the shape of helix 7, it is divided into two cylinders. This gives the model a total of 13 cylinders per monomer subunit. This model can be seen in Figure 4.

Interaction sites were defined along several cylinders in the course grained model. An NTD-NTD interface is defined between cylinders 2 and 3. Values used to determine the potential were $\sigma = 0.9702867$ nm and $\theta_0 = 0.690483$. An NTD-CTD interface is defined between cylinders 4 and 10, corresponding to helices 4 and 8, with $\sigma = 0.7257217$ nm and $\theta_0 = 1.218533$. A CTD-

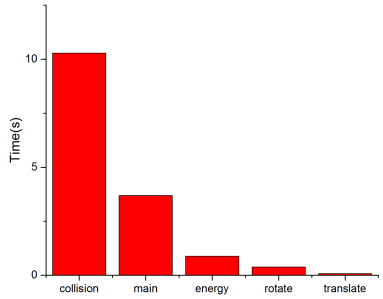

Figure 3: The breakdown of time spent on algorithms during a short simulation.

CTD trimerization interface is defined between along cylinder 12, corresponding to helix 10, with $\sigma = 0.982183$ nm and $\theta_0 = 0.5061$. For cases where the monomer was used as the most basic subunit, the dimerization interface was defined along cylinder 11, corresponding to helix 9, with $\sigma = 0.9841567$ nm and $\theta_0 = 0.737260761$. In most cases, the ϵ_0 can be taken to equal 4 kcal/mol or 6 kcal/mol in the case of the dimerization interface, but this value can be modulated to see how it affects the system[28].

Hypotheses

It is hypothesized that the simulation should allow subunits to assemble into a lattice resembling the authentic HIV-1 capsid. From this, the curvature and arrangement of oligomers can be determined. It is also predicted that as the interface strengths are varied, the curvature of the lattice will change. This can help to identify the significance of interfaces in the capsid assembly.

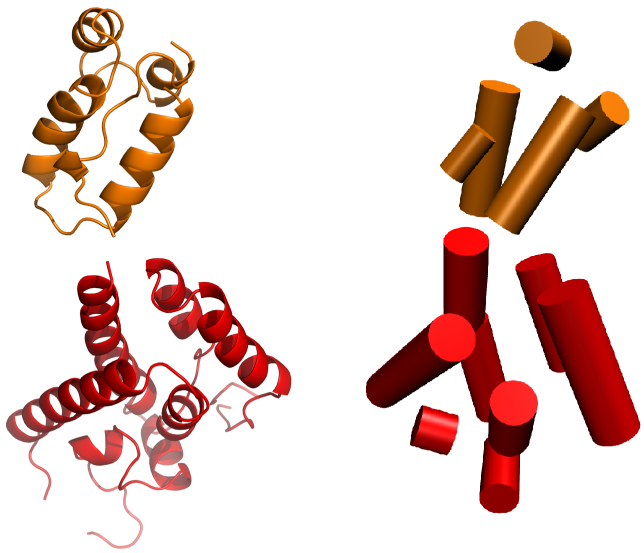

Figure 4: The actual capsid protein(left) compared to the model(right). Figure adapted from Chen and Tycko[28].

It can also be said that the significance of the dimerization to the assembly process can be determined by using the monomer as the most basic subunit. Dimerization is believed to be important in the correct and efficient assembly of the HIV-1 capsid.

OPTIMIZATIONS

Verlet List

A verlet list is designed to reduce the computational cost of the collision detection and energy calculation algorithms by storing the closest neighbors to a given subunit and only considering those neighbors for the algorithms. In order to create this list, an algorithm was constructed that computed the distance between the centers of subunits. If the subunits were close enough, they would be added to each others list. The distance chosen depends on the distance at which we define the cutoff of intermolecular forces, 10.91 nm, and the number of steps that must be performed before the list needs to be updated again. This step size was chosen to be 50, meaning that if two molecules moving at maximum translational steps of 0.05 nm for 50 steps they would each be able to move 2.5 nm in any Cartesian direction, and in the worst case they could move 5.0 nm closer to one another. For this reason, the cutoff limit for subunits to be included in the verlet list was calculated using

$$\text{verlet cutoff} = (\text{force cutoff}) + 2 \times (\text{step size}) \times (\text{translation step}). \tag{7}$$

In order to minimize the number of times that the verlet list needed to be reset, the criteria was changed from updating every 50 steps to updating each time any subunit was displaced more than 2.5 nm, for reasons given above, from where it was the last time the verlet list was set. This was accomplished by storing the old positions when the verlet list was reset the last time and comparing it to the current location on steps where the molecular move was accepted by the metropolis criterion. If the move was rejected, this check can be avoided since the subunit will remain at the same location from the previous step.

The size of the verlet list can be set at the beginning of the simulation by determining the maximum number of subunits that can be within the verlet cutoff distance. This is done by calculating the volume of the sphere created by using the cutoff distance as the radius and dividing this volume by the volume of a cylinder that encloses a single subunit. This should give the theoretical maximum number of subunits that can be included within a particular subunit's verlet list.

The implementation of a verlet list requires that the collision detection and energy calculation algorithms be modified to accept a verlet list. While presumably the omitted subunits would trigger the best case runtimes of the algorithms as described in the methods, there are still significantly fewer comparisons during each call, therefore improving the speed of the simulation. The improvement in simulation efficiency after the implementation of a verlet list can be found in Table 2.

By Dimer

One way of defining the distance between subunits is from the center of the dimers, which is the rotational center of the subunits in the simulations. The verlet list would store dimers that are within the verlet cutoff distance. The advantages are that the coordinates of the rotational center are not effected by the rotation of the subunit. The drawbacks are that the cutoff limit for intermolecular forces as well as the collision detection and energy calculation algorithms are designed for monomers. This means that the list may include entries that needed only by one or neither monomer in the dimer.

By Monomer

Another way of defining the distances between subunits is from the center of each monomer. The verlet list would have to be larger in terms of space to accommodate the fact that there are more monomers in the system, but would overall contain fewer subunits since each entry represents one

monomer instead of two. A possible drawback is that due to molecular rotation, it is possible for a monomer to travel slightly faster than the 0.05 nm per step, but this is easily offset by using a distance check instead of a set number of steps, which was decided upon regardless. The advantages are that the algorithms and necessary constants are designed for monomers.

Overall, it was found that verlet list by monomer was slightly faster than verlet list by dimer. Therefore it was decided that the verlet list by monomer would be used going forward.

Sorting the Verlet List

When the verlet list is set, it is possible to order the entries from furthest to closest in the verlet list. While this provides no clear advantage in the energy calculation algorithm, it can increase the speed of the collision detection algorithm. Closer subunits are more likely to incur collisions, and by testing them first it is more likely that the algorithm will be able to exit early due to a collision. Especially when subunits are close together, this can save a lot of computational time.

Removing the Interfaces

In early versions of the code, the location of interfaces used in energy calculation was stored in a manner similar to the location of helices where they had to be translated and rotated during each step before any calculations were run. By instead calculating the interface location based on helix location within the energy algorithm, it is possible to improve the speed of the simulation.

Table 2: Improvement in performance after the implementation of a verlet list.

code	total steps	accepted steps	improvement(total)	improvement(accepted)
Original	573,922	188,625	N/A	N/A
Verlet List	944,947	319,403	1.646	1.693
Sorted Verlet List	976,433	626,882	1.701	1.733

This was done by defining the distance along a helix that one must travel to reach the location of the interface. This could be done for both interacting subunits and the distance between interfaces could then be calculated. The advantage is that additional rotation and translation calculations can be avoided, and in the case of collisions these calculations could be cut out in whole. In cases where there is no collision, this method provides an accurate way of determining interface location that can be used in the energy calculation algorithm.

Compiler Optimizations

Most compilers provide optimization options that may affect the efficiency of programmed algorithms. Some may do this by default, and others require flags to trigger optimizations. Care must be taken when using these optimizations, however, as there is the possibility that the code may not behave as intended. For this reason, the simulation must be checked to ensure that the compiler optimizations did not degrade the programs integrity.

GNU Fortran Compiler

The GNU compiler does not by default optimize the code. This was the method used to compile early versions of the code, and provided accurate, yet inefficient assemblies.

Compiler Flags

When the code is compiled in a Linux environment using the `-O3` compilation flag, it provides the highest level of optimization that is supported by the GNU compiler. Although this is not the best for debugging, the optimized code runs significantly quicker than the unoptimized code. Runtime analysis for the GNU compiler can be found in Figure 3.

Intel Fortran Compiler

Unlike the GNU compiler, the Intel compiler optimized the code without any compilation flags. This means that by default, the Intel compiler will be faster than the GNU compiler when no compilation flags are used.

Compiler Flags

In order to properly debug the program, optimizations must be removed. This can be done by using the `-O0` flag when compiling the code. This provides the purest, yet slowest, version of the code. In cases where optimization was permitted, the `-O2` compilation flag was used. This represents the highest level of optimization provided by the Intel compiler. The Intel compiler with the `-O2` flag was found to be the fastest compiler and optimization combination. In environments where the Intel compiler is available, the Intel compiler will be used with the `-O2` flag. Runtime analysis for the Intel compiler can be found in Figure 3.

OpenMP

When used with supported compilers, OpenMP provides the programmer a way to parallelize their code using a set of flags that appear as comments when OpenMP is not used, but provide directives when OpenMP is enabled. OpenMP's flagging system provides an easy way to

Table 3: Comparison of Fortran compilers and compiler optimizations. Data was averaged over five runs on initially identical systems. Note that OpenMP was enabled during tests.

compiler	flag	time(avg)
gfortran	none	29.223
gfortran	-O3	14.306
ifort	-O0	35.551
ifort	-O2	10.648

tell the compiler about which information is shared and how, and leaves the compiler to parallelize the flagged sections of code. OpenMP can be used in conjunction with compiler optimizations, and can greatly improve the speed of certain algorithms. Parallelization works particularly well with repetitive tasks that do not need to exchange information. Improvements over base code can be seen in Table 4.

Algorithms

In order to test the potential gains of a simulation optimized using OpenMP, some major algorithms were targeted for tests. Tests were performed by isolating the algorithm and running it on a predefined system a large number of times. The three major algorithms that were targeted for improvement by OpenMP were collision detection, energy calculation, and the setting of the verlet list. Other algorithms, such as the rotational and translational components of the code, were not work intensive enough to benefit from OpenMP parallelization.

Collision Detection

The collision detection algorithm was parallelized using an `OMP DO` loop. This loop was tested on various levels of the collision detection algorithm, but was found to run most effectively when placed on the uppermost tier of the algorithm, the loop over subunits in the verlet list. It was also determined that scheduling this loop dynamically provided a bigger improvement than when scheduled statically. Dynamic scheduling means that a new task is assigned as the previous task is completed instead of dividing the work into chunks beforehand. This improvement may be due to the distribution of unequal distribution of subunits to each thread. If one thread ends up with many nearby subunits while another thread ends up with further subunits, then all threads must wait for the slowest thread to complete. This may be worsened by the the sorting of the verlet list.

In order for the algorithm to retain its ability to finish quickly if a collision is detected, an `OMP ATOMIC` command was used to synchronize the threads. OpenMP does not allow the loop

to close early, but by allowing the loop to cycle when a collision is detected, most, if not all, work was removed during each iteration, allowing the algorithm to finish almost immediately.

Energy Calculation

The energy calculation algorithm was tested using several methods of parallelization. It was first parallelized using an `OMP DO` loop over the verlet list as was found to be the most effective implementation in the collision detection algorithm. Unfortunately, the cost to set up the loop actually outweighed the speed gains of the parallelization. This may be due to the relatively little work required for the energy calculation algorithm when compared to the collision detection.

Although the `OMP DO` loop was found to decrease performance, it was found that be using an `OMP SECTIONS` in the main code dividing up the first and second monomers within the dimer could modestly improve performance.

Verlet List

The verlet list creation algorithm was also parallelized using an `OMP DO` loop on the outermost loop. It was found that this greatly improved performance, even to a greater degree than the collision detection algorithm. This can be attributed to the nested loops and relatively straightforward design of the setup process. Unlike the collision detection algorithm, information did not need to be shared between threads and there was no need for an ability to exit early.

sections_7-16

In order to try to increase the workload of each thread, therefore maximizing the payout of OpenMP implementation, all collision detection and energy calculation was done within one large `OMP DO` loop. This do loop contained many `OMP ATOMIC` calls to synchronize energy and collision data. If a collision was detected, it would skip all work in a manner similar to that described in the OMP implementation of the collision detection algorithm. Until this point, data

was recorded to file every 60 seconds. Due to the nature of parallelization, computation time was no longer equivalent to real time. To try to decrease the number of writes to file, the system was set to write only after 60 seconds multiplied by the number of threads used.

openmp_7-24

In order to compare subunits accurately, the coordinates must be corrected for periodic boundary conditions before collision detection or energy calculation algorithms are run. It was predicted that by only calculating these temporary coordinates once to be used for both collision detection and energy calculation of the displaced molecule, computational time could be saved. Due to the differing positions, the temporary coordinates of the original location must be computed.

openmp_7-26

This modification did not actually involve any modification to the OpenMP portion of the code. A condition was added to the collision detection algorithm to avoid calculations if certain conditions were met. This was performed on the lowest level of the algorithm where individual cylinders were compared. It was added that if the lines representing the axes of the cylinders were far enough apart, the code would not go through the trouble of collision detection along the entire cylinder.

Until this point, the simulation was set to record the current state of the system based on how much time has passed. Due to the nature of parallelization, one actual minute is longer than one minute of computational time. In earlier parallelized versions of the code, this lead to an incredible amount of writes that waste memory and slow down the simulation. Even after multiplying by the number of threads, the number of writes was still found to be too high. To correct for this, the simulation was set to record after 2,000,000 accepted steps instead of a set period of time. In movie creation, this can also lead to a more accurate representation of how the system forms.

openmp_7-29

In a significant shift in the ordering of the code, the energy calculation was pulled out from the `OMP DO` loop and placed in an unparallelized region before collision detection was performed. It is known that the collision detection takes significantly longer than the energy calculation, so by checking the energy criterion prior to the collision criterion, the lengthy collision algorithm can be avoided altogether in the case of energy criterion failure.

openmp_8-2

In order to further improve on the 7-29 code, the energy calculation algorithm was placed within an `OMP SECTIONS` blocks, where one section computed the energy after the move and one computed energy before the move. By organizing the sections in terms of before and after the step instead of by monomer, exchange of information between threads can be avoided.

openmp_8-20

It was believed by nesting the `OMP DO` loop in a section parallel to the energy calculation, it would allow the collision detection to begin before the energy calculation had finished. Unfortunately, OpenMP does not support this nested functionality, and allowing an unparallelized collision detection algorithm to begin early does not increase the speed of the overall simulation.

openmp_8-22

This version of the code did not take steps to improve the efficiency of the code, but instead provided a way to better continue previous simulations. Until this point, a simulation could be continued by generating a file containing the full cylindrical coordinate representation of the system and allowing the simulation to read in that file. From this file, the simulation would calculate the boundaries of the system based on the positions of the subunits. This could lead to slight dis-

continuities when continuing a simulation. The modification writes the boundaries to a file when starting a new simulation and allows the simulation to read in a boundary data file when continuing a simulation allow better continuity when resuming a previous simulation.

Message Passing Interface

Message Passing Interface, or MPI, is another way of parallelizing code. Unlike OpenMP, MPI requires direct coding of how information and tasks are distributed between threads. Algorithms were coded in MPI and compared to how they performed without MPI and with OpenMP. It was found that OpenMP performed considerably better than similar MPI algorithms. It is believed that OpenMP's ability to dynamically schedule tasks and share information that gave it this advantage. It was determined that OpenMP versions of the code would be used in future simulations.

Table 4: Comparison of OpenMP codes. Note that tests were done on a quad core processor.

threads	default	1	2	4	8
Original	13.00	N/A	N/A	N/A	N/A
sections_7-16	13.00	13.82	13.18	11.64	15.05
openmp_7-24	12.09	13.44	10.01	8.07	12.00
openmp_7-26	12.14	13.36	10.08	7.94	12.07
openmp_7-29	9.76	10.24	8.52	7.90	13.56
openmp_8-2	9.87	10.52	7.39	6.43	10.14

FINDING THE OPTIMAL INPUT PARAMETERS

Varying Interaction Strengths and Initial Spacing

The interaction strength is important in the efficiency of the simulation. As was discussed earlier, if the interaction strength is too strong, the assembly may get stuck and cease to evolve beyond a certain point. If the interaction strength is too weak, the lattice will not form. The optimal assembly will occur when subunits interact, but interactions are reversible. In order to gauge these strengths, all interaction sites were set to the same interaction strength.

Although spacing is not a huge factor in the assembly process, it is used to determine boundary and initial conditions. If the subunits begin too cramped, it may affect the assembly process. If the subunits begin too spaced out, the assembly may occur very slowly or not at all, in which case the simulation is diffusion limited. These runs were performed before compilation optimizations were performed, leading to slower assembly speed than later versions of the code. Simulations were allowed to run for 120 hours.

3.5 kcal/mol

Five runs were performed on a system of 105 dimers where the interaction strength was set to 3.5 kcal/mol. Setting the interaction strength to 3.5 kcal/mol was one of the cases where the interaction strength was too weak to form. After the simulation was allowed to run for a long time, the system still appeared completely random with no lattice formation.

4.0 kcal/mol

Ten runs on a system of 216 dimers with an interaction strength of 4.0 kcal/mol were performed. Half of these runs had an initial spacing of 10 nm and the other half had an initial spacing of 7.8 nm.

10 nm spacing

Subunits appear to be assembling, with several groupings of interacting dimers. The early stages of lattice formation can be seen, where some hexamers have been formed and other oligomers appear to be collecting around them. The system can be seen in Figure 5.

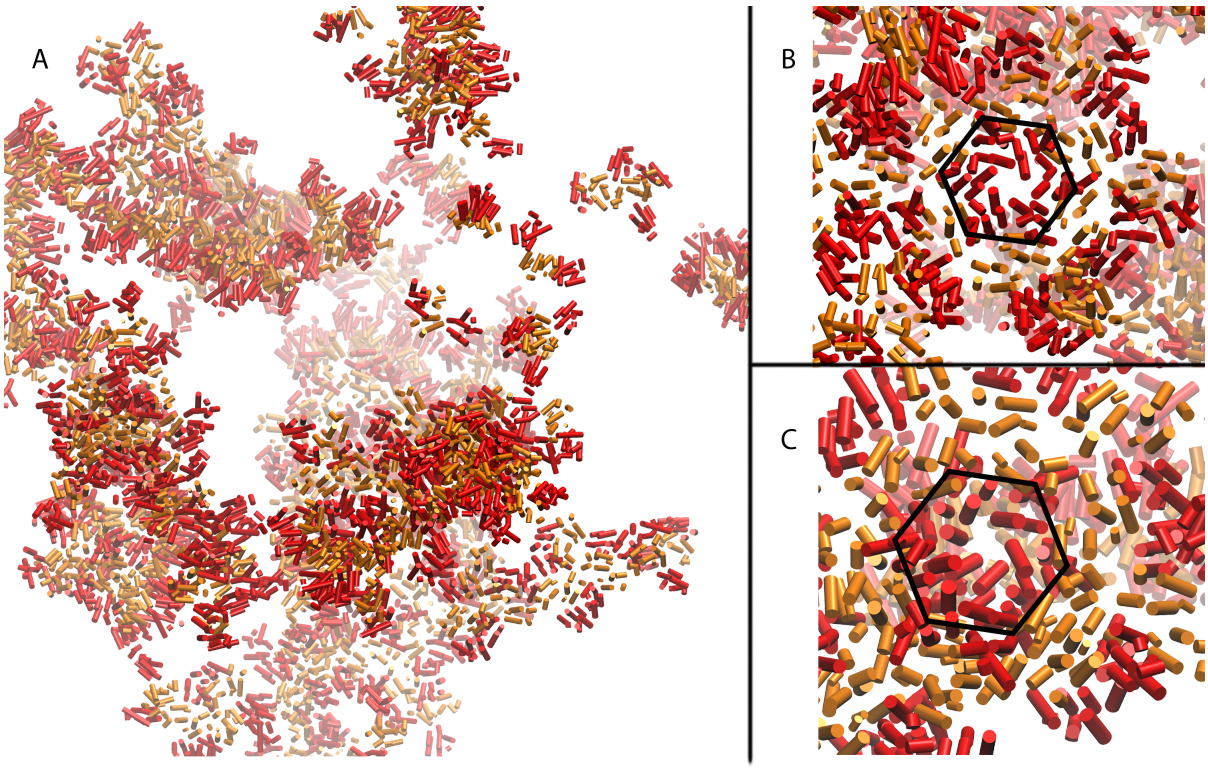

Figure 5: The results of a simulation started with 4.0 kcal/mol interaction strengths and 10 nm initial spacing. (A) An overall view of a system. (B,C) Hexamer formation and the beginning of lattice formation in the system.

7.8 nm spacing

Although subunit formation was observed, the assembly appeared cluttered. Subunit formations appeared to be right next to each other, but not in a manner that would support lattice formation. Several larger oligomers were observed in these systems. A system can be seen in figure 6.

5.0 kcal/mol

Ten runs on a system of 216 dimers with an interaction strength of 5.0 kcal/mol were performed. Half of these runs had an initial spacing of 10 nm and the other half had an initial spacing of 7.8 nm.

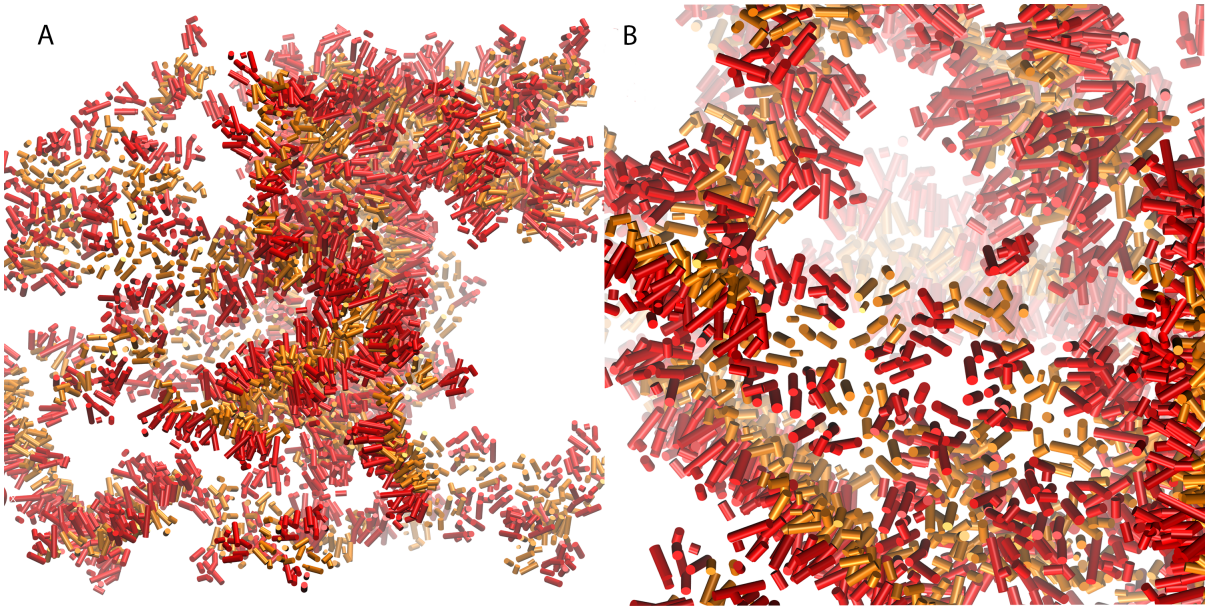

Figure 6: The results of a simulation started with 4.0 kcal/mol interaction strengths and 7.8 nm initial spacing. (A) A broad overview of the system. (B) Even within the cluttered assembly, some larger oligomers, including hexamers, can be observed.

10 nm spacing

Larger oligomers surrounded by other subunits were common, and the system appears to be in the very early stages of lattice formation. Formation was not observed to be significantly different than that of the 4 kcal/mol runs(Figure 7).

7.8 nm spacing

Similar to the 4.0 kcal/mol interaction strength with the same initial spacing, the assembly appeared cluttered. Formations appeared clustered, but further oligomer formation was observed. Early lattice formation did not seem as apparent as the 5.0 kcal/mol runs with 10 nm spacing(Figure 7).

6.0 kcal/mol

Ten runs on a system of 216 dimers with an interaction strength of 6.0 kcal/mol were performed. Half of these runs had an initial spacing of 10 nm and the other half had an initial

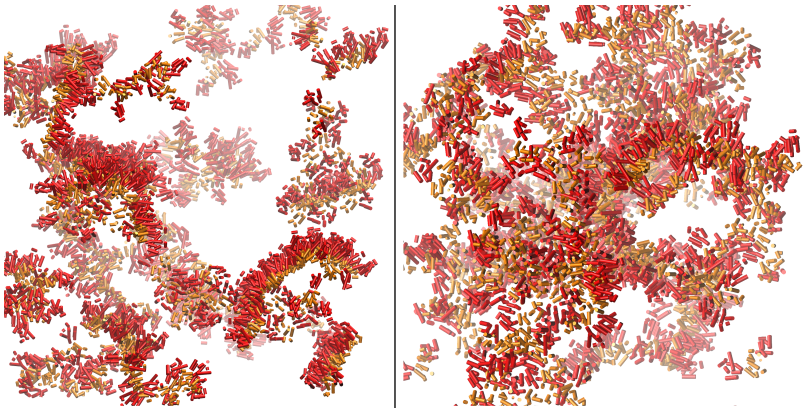

Figure 7: The results of a simulation started with 5.0 kcal/mol interaction strengths. (A) A system with an initial spacing of 7.8 nm and (B) a system with an initial spacing of 10 nm. There is no clear pentamer, hexamer, or lattice formation.

spacing of 7.8 nm.

10 nm spacing

No lattice formation was observed. Large oligomers were sparse in the assembly, and did not appear to be forming as in the 4.0 and 5.0 kcal/mol interaction strength runs(Figure 8).

7.8 nm spacing

Formations similar to the 4.0 interaction strength with the same spacing were observed. A cluttered, chained assembly with minimal to no large oligomers forming. Although cluttered in certain areas, there were also more gaps in the system then what was observed at lower interaction strengths with the same initial spacing(Figure 8).

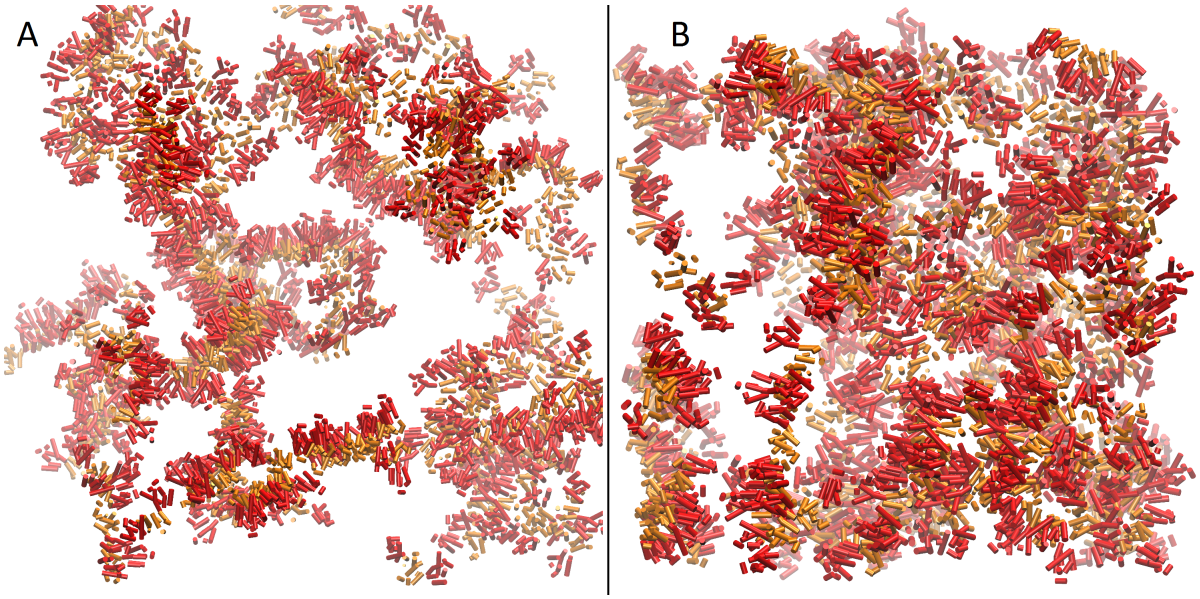

Figure 8: The results of a simulation started with 6.0 kcal/mol interaction strengths. (A)A run with 10 nm initial spacing and (B) a run with 7.8 nm initial spacing. Collections of subunits can be seen, but there was no observed large oligomers or lattice formation.

Larger spacings

Following these tests, runs were set up to test initial spacings of 12 nm and 15 nm with 4.0 kcal/mol strengths. The systems tested were smaller at only 60 dimeric subunits per simulation. It was found that the 12 nm would produce assemblies similar to the 10 nm spacing, but with a higher rate of assembly failure. At 15 nm, the assembly would consistently fail. Significantly more MC steps would occur, but there would be no oligomer or lattice assembly(Figure 9). This implies that the spacing was too great for the assembly to occur.

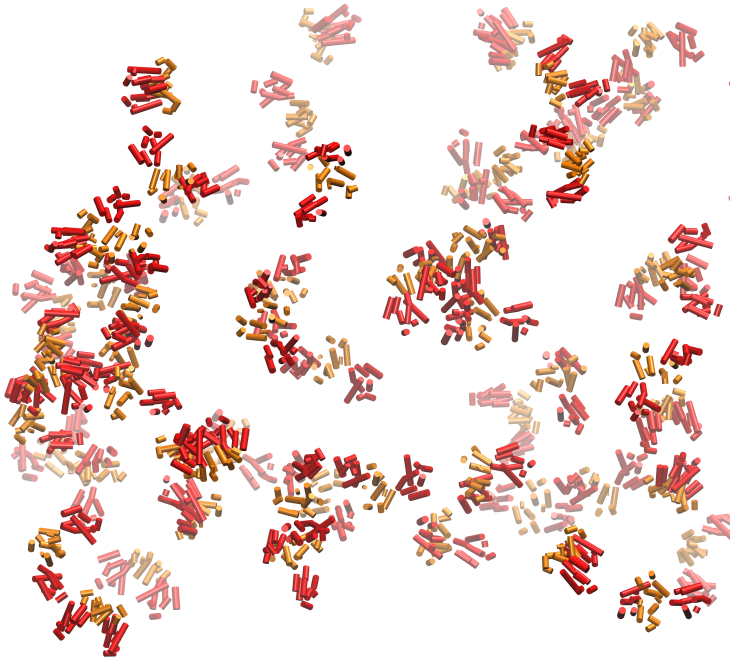

Figure 9: The results of a simulation started with 4.0 kcal/mol interaction strengths and 15 nm initial spacing. A snapshot was taken after the same amount of time had passed as with other tests. No observable oligomer or lattice formation was present in the system.

Conclusions

Based on these runs, it was determined that a 4.0 kcal/mol interaction strength with 10 nm initial spacing would be used moving forward. Based on the runs observed, the 7.8 nm space produced an assembly that was too cluttered to form correctly, with the 10 nm spacing providing a better visualization of the system. The 4.0 kcal/mol interaction strength also produced the best, although early, lattice formation. This interaction strength was stronger than that found by Chen and Tycko[28]. This may be due to the additional degrees of freedom in three as opposed to two dimensional space. This number is also similar to the value given in a paper by Grime and Voth[25]. It is also within the interaction range given by Ceres and Zlotnick for the Hepatitis B viral capsid[30].

System Size

In order to obtain the greatest amount of information in the least amount of time, an optimal system size must be determined. Although MC simulations should always find the lowest energy state if given enough time, it is much more useful to observe lattice formation over the course of a few days instead of a few weeks. In order to determine the range of optimal system sizes, four system sizes were chosen. The chosen sizes were 60, 128, 175, and 256 dimeric subunits. Each system size was repeated three times for 240 hours each.

60 Dimers

All runs managed to form into one coherent lattice in the allotted time(Figure 10). The first run managed to form into a single lattice fairly quickly, but even after being allowed to run for the full time did not develop any significant curvature. The second system took about the same amount of time as the first to form into a single, flat lattice. Unlike the first system, the second managed to form curvature. After being allowed to run for the full simulation time, there

is curvature apparent within the lattice. The third run also produced interesting results. After a short amount of time, the system managed to form a flat lattice. In about the same amount of time it took the other simulations to form a flat lattice, the third system began to form curvature. This curvature persisted until the end of the simulation. Curvature data for these systems can be found in Table 5.

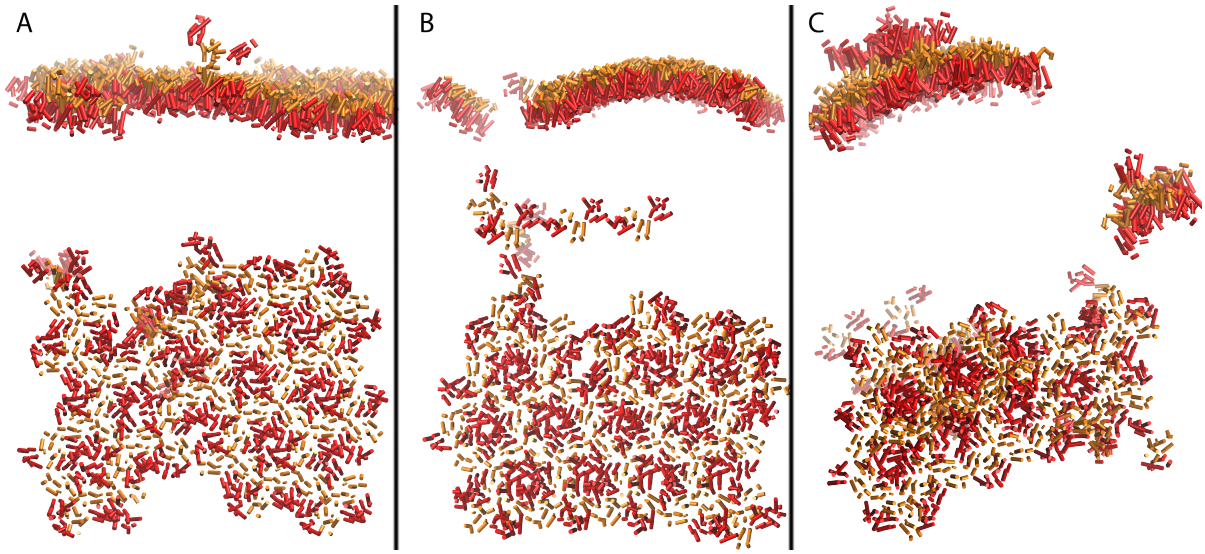

Figure 10: Three simulations run with 60 dimeric subunits. A side profile and top down view were taken for each of the three simulations. (A) A flat lattice formed and persisted throughout the assembly. (B,C) Curved assemblies were also present, and evidence that 60 subunits could form a single, coherent lattice in a relatively short amount of time was observed. Note that discontinuities in lattices can be attributed to the periodic boundary conditions.

Table 5: Curvature of a system of 60 dimers. The average over several hundred steps was used to determine an average radius.

	1	2	3
Radius (nm)	109.1	78.9	87.4

Such a small system appears to have several advantages. The first and most noticeable is that the system forms a single lattice fairly quickly. Although larger simulations can produce lattices greater in size than possible with a system of only 60 dimers, it is more likely that the larger simulations will produce other formations that are scattered from the main assembly. The benefit of such a small system is that it increases the probability that one, full connected lattice will form. This also makes the system easier to visualize, allowing for a more concentrated focus. This is beneficial when checking the consistency and affects of modified parameters.

128 Dimers

One run produced two about equally sized sheets, which appeared near each other, but facing opposite directions by the end of the simulation. Neither sheet showed any sign of curvature. This visual provides a good example of why larger system sizes may take significantly longer to form. In order for these two lattices to eventually come together, they must combine to rotate 180°. The number of MC steps to accomplish this task with two sheets of this size can be very large. The next run produced two main formations, one large sheet and another lattice of about four hexamers. At some points during the simulation, the sheet appeared to develop some curvature. At the end of the simulation, the large sheet appeared to form a twisted curvature, as if one end had been held fixed and the other twisted. Although gradual, this twist is evident. The last run did not produce anything significant.

It appears that the 128 system size did not always form as desired(Figure 11). The system size did not appear too large for the simulation to handle, but from what was observed, it could take many more steps before the systems form into single lattices, and even longer to show fixed curvature.

175 Dimers

Two of the runs resulted in one large sheet being created with several smaller formations throughout the system. Although these lattices were well formed, only one of them showed evidence of curvature. The other run did not produce any significant results, but did show signs of lattice formation. These findings imply that 175 is not an optimal system size for this simulation(Figure 12).

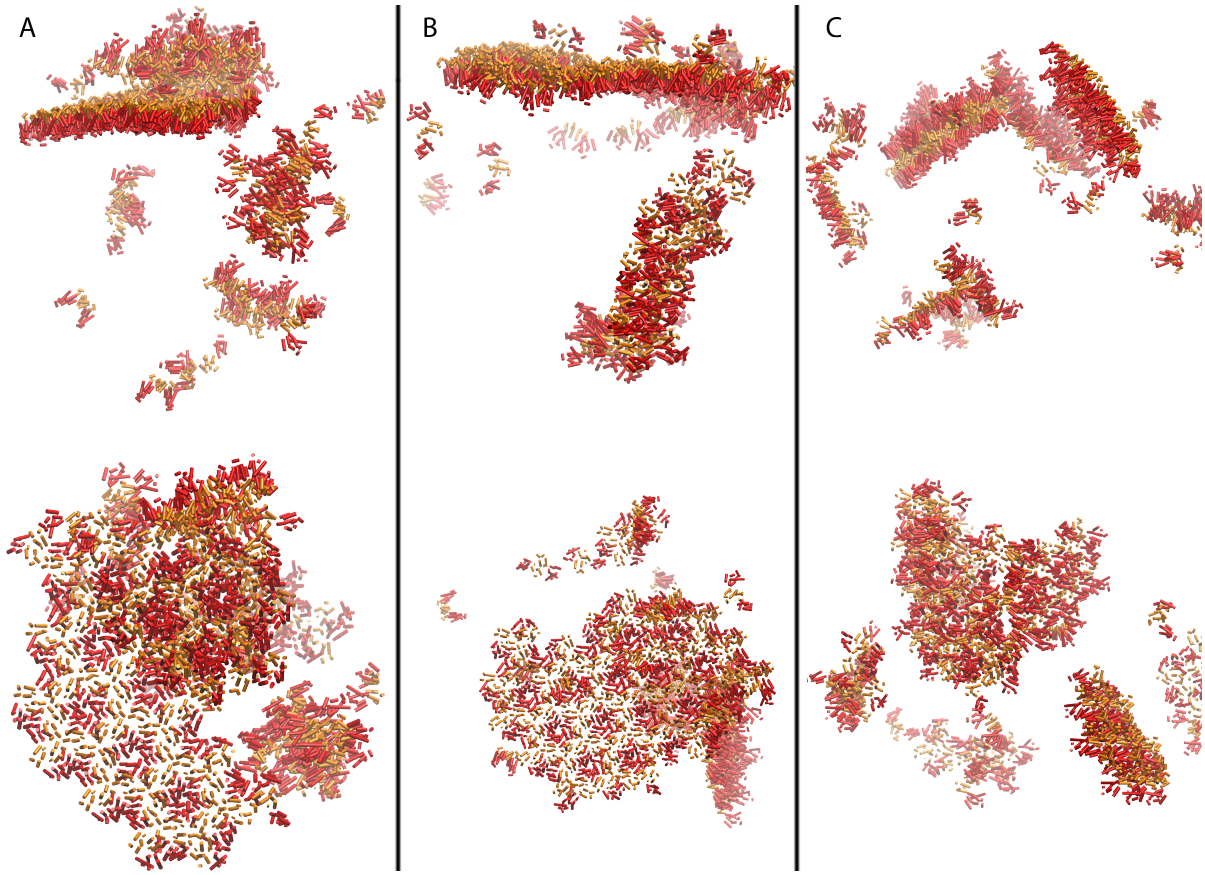

Figure 11: Three simulations run with 128 dimeric subunits. A side profile and top down view were taken for each of the three simulations. (A) Two main sheets can be seen in this assembly. The side profile gives a good idea of how they face opposite directions and the top down view shows their relative positions. (B) A twisted assembly can be seen by looking at the side view. (C) A small, double layered assembly can be seen, but with no significant lattice formation.

256 Dimers

The first run produced one main sheet and several other large sheets. The main lattice and several others appeared to curve slightly. The other two runs did not produce any significant results. This appears to be another case where the system size is too large to see good formation in a reasonable amount of time. The advantage of a system this large is that several lattices can be generated, as seen in the first run. Otherwise, this system size shows no direct benefits over smaller system sizes.

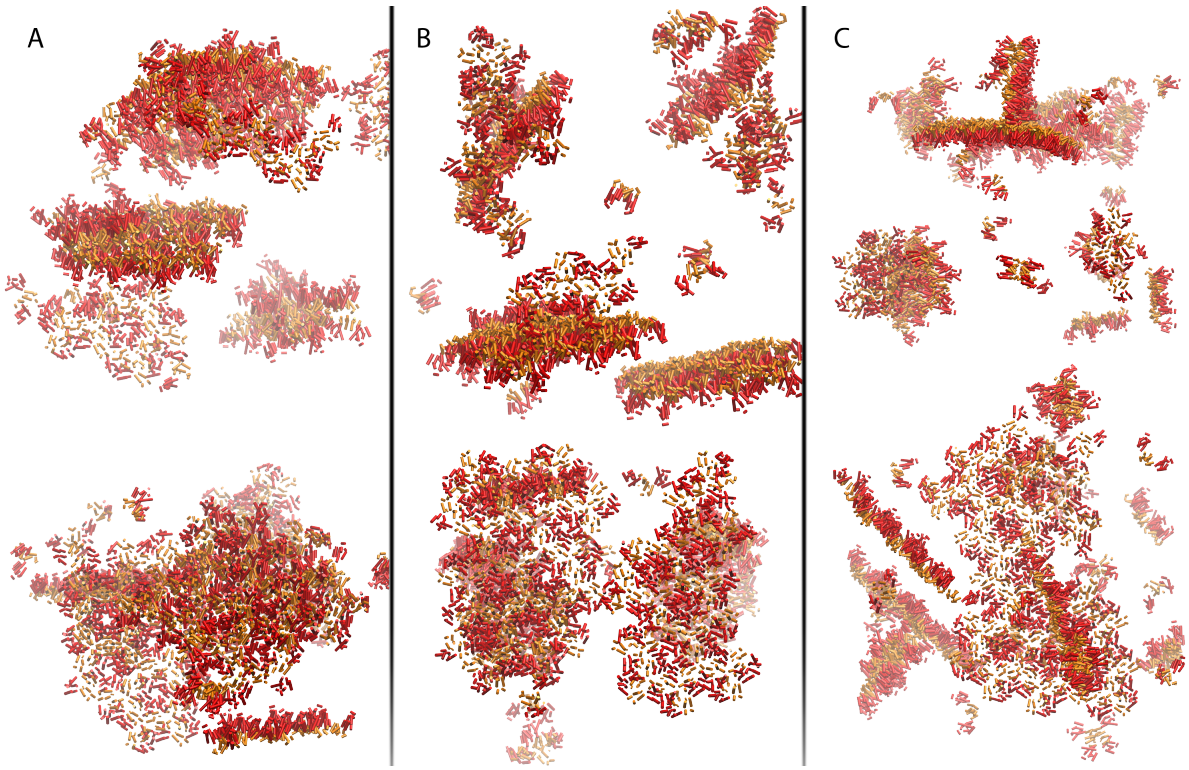

Figure 12: Three simulations run with 175 dimeric subunits. A side profile and top down view were taken for each of the three simulations. (A,B) Signs of lattice formation are apparent throughout the system of 175 dimers. (C) Curvature was apparent in only one simulation of 175 subunits.

Conclusions

It appears that smaller system sizes of under 100 subunits provide for more conclusive assemblies. In order to see how changing parameters affects lattice formation, it is easier and more consistent to observe these smaller systems than larger systems. Systems of this size seem to complete within a reasonable amount of time, avoiding the need for longer simulations and reducing the chance of unexpected lattice formation.

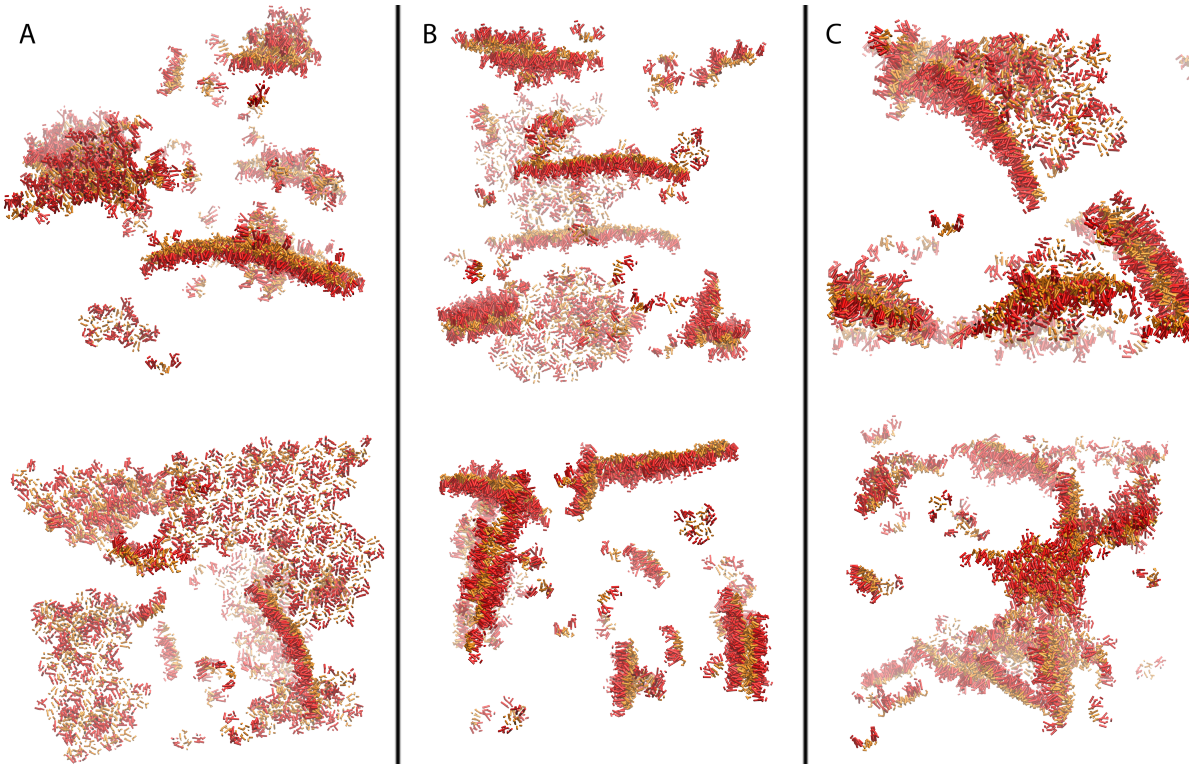

Figure 13: Three simulations run with 256 dimeric subunits. A side profile and top down view were taken for each of the three simulations. (A,B,C) Signs of lattice formation were evident across all simulations, but no simulation managed to form a single, coherent lattice.

Flexibility of Interfaces

It was believed that by adding a distance and/or angular flexibility to the energy criteria, more curvature could be observed. This was done by

$$\theta = \begin{cases} \theta_0 & |\theta - \theta_0| \leq 5^\circ \\ \theta & |\theta - \theta_0| > 5^\circ \end{cases} \tag{8}$$

where θ_0 is the optimal angle and θ is the observed angle and

$$d = \begin{cases} d_0 & |d^2 - d_0^2| \leq 0.4 \times d_0 \\ d & \text{otherwise} \end{cases} \tag{9}$$

where d_0 is the optimal distance and d is the actual distance. Runs were set up where these criteria were used together and separately. The runs encompassed all interfaces together and separately. It was found that these criteria did not improve the assembly and in many cases impeded the assembly. It was also found that by allowing flexibility, the consistency of assemblies was decreased. It was decided that this flexibility would not be used in future simulations.

Trimeric Interface

The trimeric interface in this model is defined between cylinder 12 of two or more subunits. It is important in bonding between two or more hexameric subunits. It was thought that by strengthening the interface, a more consistent curvature could be observed. For this reason, the interface strength was varied to 3.0, 3.5, and 5.0 kcal/mol to see how the assembly would be affected. Several runs were set up where system sizes were set to 60 subunits and initial spacing to 10 nm.

3.0 kcal/mol

Weakening of the trimeric interface did not hinder the development of hexameric lattices. As before, the subunits were able to form hexameric oligomers and assemble into a sheet(Figure 14). An interesting development was that throughout the assembly, no double-layering of sheets was observed. This provides evidence that the trimeric interface is responsible for the double-layered formations observed in earlier trials. The reduction in strength of the trimeric interface to 3.0 kcal/mol did not seem to affect the consistency of the curvature of the assembly, as seen in Table 6.

Table 6: Curvature of a system of 60 dimers where the trimeric interface strength was set to 3.0 kcal/mol. The average over several hundred steps was used to determine an average radius.

	1	2	3	4	5	6	7	8	9	10
Radius (nm)	85.4	52.6	115.1	126.3	126.1	90.6	126.6	86.8	52.7	86.8

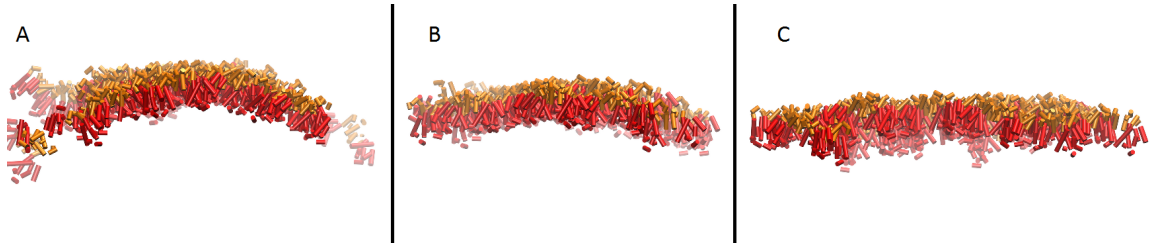

Figure 14: The systems observed when the trimeric interface strength was changed to 3.0 kcal/mol can be seen. (A) Some assemblies showed signs of sharp curvature, (B) while others appeared with very mild curvature or (C) no curvature at all. Double-layered assemblies were not observed when the interaction strength of the trimeric interface was this weak.

3.5 kcal/mol

As seen in trials where the trimeric interface had a 3.0 kcal/mol interaction strength, hexameric lattices were observed when the strength was set to 3.5 kcal/mol. Double layering was observed in a few runs, but was less prevalent than at higher strengths. This trimeric interface strength produced mostly flat assemblies, but still does not yield a consistent final curvature. Some systems can be seen in Figure 15.

5.0 kcal/mol

Strengthening the trimeric interface did not produce a correct assembly. While it appeared that some typical oligomers, such as hexamers, were formed during the assembly, the lattice formed incorrectly. It was found that the strengthening of the interface produced a clumped assembly due to the domination of the trimeric interface.

Conclusions

While modulating the strength of the trimeric interface did not yield a consistent curvature, it provided more evidence for the cause of double-layering of hexameric sheets. It also shows

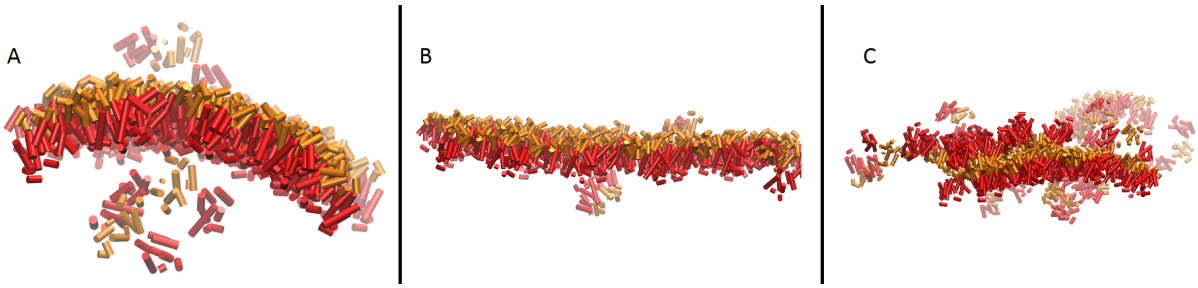

Figure 15: The systems observed when the trimeric interface strength was changed to 3.5 kcal/mol can be seen. Formations similar to the 4.0 kcal/mol strength were seen. (A) Some assemblies showed signs of curvature, (B) but other assemblies appeared very flat. (C) There were also systems where the double-layered structure was observed.

that strengthening the trimeric interface causes incorrect assembly of the lattice. This information could prove useful moving forward.

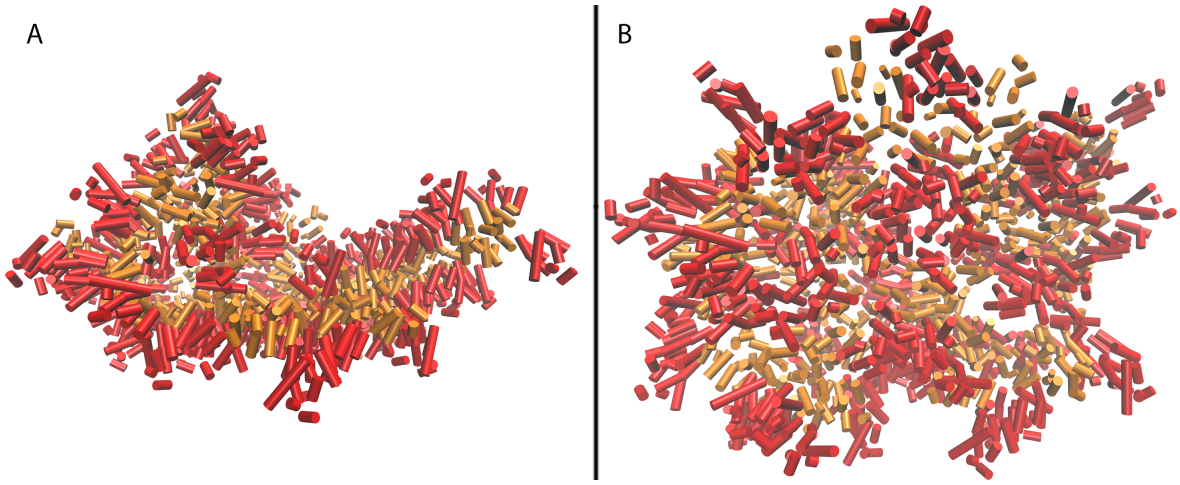

Figure 16: The systems observed when the trimeric interface strength was increased can be seen here. (A) From the side it appears to have some curvature, (B) but seen from a different angle it is apparent the lattice forms incorrectly.

MONOMERIC SUBUNITS

Code Modifications

The base code had to be changed in order to accept the capsid protein in its monomeric form. Fortunately, many subroutines were already set up to analyze each monomer in the dimer, so many adaptations were as simple as having algorithms check a single monomer instead of checking two monomers in a dimer.

Main Code

The main code had to be changed to loop over a system of monomers instead of a system of dimers. Previously, the code had implemented a counter that reset when the end of the system was reached in a manner that allowed it to loop over all dimers in the system. This mechanism was replaced by an additional loop over all monomers in the system.

Verlet List

The verlet list was previously set up to store adjacent molecules indexed by monomer. This made the modifications to the verlet list algorithm relatively simple. The major change that had to be made was the inclusion of all monomers in the system for consideration. Previously, there was no necessity to include the paired monomer as the dimer was static, making collision and energy detection between the two irrelevant. In order to change the code to treat monomers individually, this exception had to be eliminated.

Energy Calculation

The reduction of subunits from their dimeric form to their monomeric form made it necessary to accommodate an additional dimeric interface. Previously, the energy calculation algorithm

required the location of five cylinders to determine the system energy. The inclusion of the dimeric interface made it necessary to know the location of a sixth cylinder to determine the energy of the system. The energy across the dimeric interface was calculated in the same manner as other interfaces. The initial optimal Lennard-Jones distance was set to .9841567 nm and an optimal angle of 0.737260761 radians. The dimeric energy was also initially taken to be 50% greater than the other interfaces.

Collision Detection

The collision detection algorithm did not need to be significantly revised, as it was already set to detect collisions based on each monomer. The inclusion of necessary monomers within the verlet list was enough to allow the dimeric collision detection algorithm to function the monomeric subunit. The only change came in the main code, where the collision detection algorithm only needed to be called once per subunit.

Optimizations

Optimizations applied to the dimeric version of the code carried over into the monomeric version. The verlet list and collision detection algorithms remained optimized using `OMP DO` loops, and the energy calculation algorithm remained within an `OMP SECTIONS` block. In addition to parallelization, compiler optimizations could also be used with the monomeric code.

Runs

During the assembly of the HIV-1 capsid, the capsid protein exists as a dimer in solution. This modification to the code should allow give some insight into the role and importance of the dimeric interface.

Varying Interface Strengths

While holding the other interfaces constant at 4.0 kcal/mol, the strength of the dimeric interface was varied to see if an optimal strength could be determined. What was found was rather interesting. It was found that when using these criteria for the interaction energies, the monomers would form into large clusters. These clusters somewhat resembled what could normally be observed around the trimeric interface, but in a much bulkier formation. The clusters can be seen in Figure 17. It is possible that the proximity of the dimeric and trimeric interfaces created a combined potential that was greater than what could be exerted across other interfaces. It should also be noted that the dimeric and trimeric interfaces are not limited to just one pair of helices, so the collection of monomers could indeed create very large potentials in this region. For this reason, more runs were proposed where all interfaces were varied.

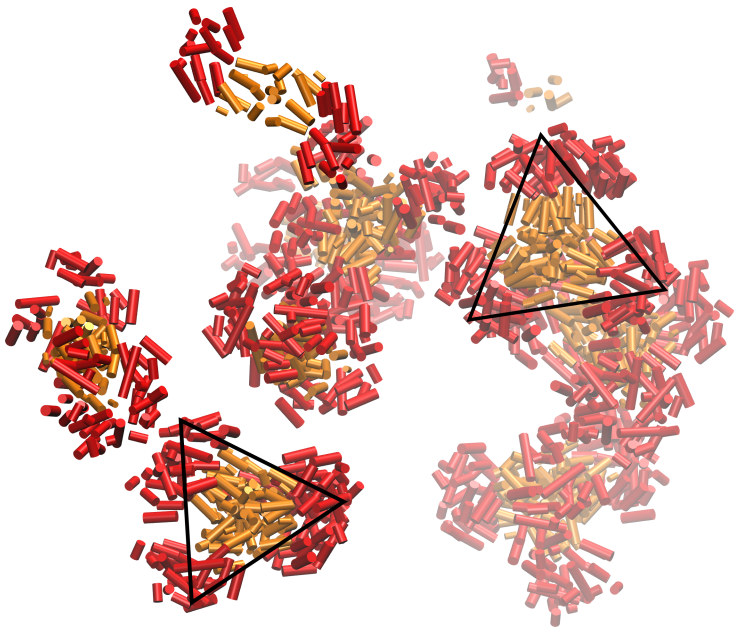

Figure 17: Clusters found while simulating monomer assembly.

Eliminating the Trimeric Interface

For this set of runs, the interactions across the trimeric interface were eliminated. If the cause was indeed the stacking of the trimeric and dimeric interface strengths, this modification would indicate that. The interaction strength of the dimeric interface was set to 6.0 kcal/mol and all other strengths were left at 4.0 kcal/mol. While this modification seemed to help eliminate the incorrect formations that were previously observed, correct dimer formation was not observed. As was observed previously, the dimerization interface did not only interact with one additional monomer. This caused small clusters to form with more than two monomers. In addition to this abnormality, the dimerization interface bonded at a sharper angle than expected. One possible explanation for this is the interaction between other interfaces on each monomer.

Eliminating Interactions between Oligomers

In the dimeric assembly, adjacent oligomers are typically connected across the dimeric and trimeric interfaces. In order to see monomers interact within an oligomer, the interaction across these interfaces was eliminated. The simulation was run for interaction strengths set to 6.0 kcal/mol and 8.0 kcal/mol. It was found that not only did these parameters produce oligomers typically found in the capsid assembly, such as hexamers and pentamers, but also larger oligomers containing as many as nine subunits. These oligomers can be seen in Figure 18. More surprisingly, it was found that during the late stages of assembly with an interaction strength of 8.0 kcal/mol, the subunits formed into spirals. These spirals can be seen in Figure 19. This is not characteristic of the expected assembly, but an interesting development in the simulation.

Varying All Interfaces

An attempt was made to balance the interaction strengths in a manner that would allow some assembly. The dimeric interface was set to 8.0 kcal/mol, the trimeric interface was set to

2.0 kcal/mol, and other interfaces were set to 5.0 kcal/mol. The resulting lattice was incorrect, but some formation was observed. The lattice appeared to form as a chain, but no standard oligomers were observed.

Varying the Dimerization Cutoff Angle

In order to try to invoke correct interaction across the dimeric interface, runs were set up where the cutoff angle was modified. It was proposed that by limiting the range by which the monomers could wander while still interacting across the dimeric interface, it would encourage a more stable dimer. This did not seem to be the case, as the formation of the dimer did not appear to be greatly impacted by these changes. It was also found that eliminating all other interactions did not eliminate the appearance of sharper angled dimers, implying that other interactions were not the culprit in the malformed dimers.

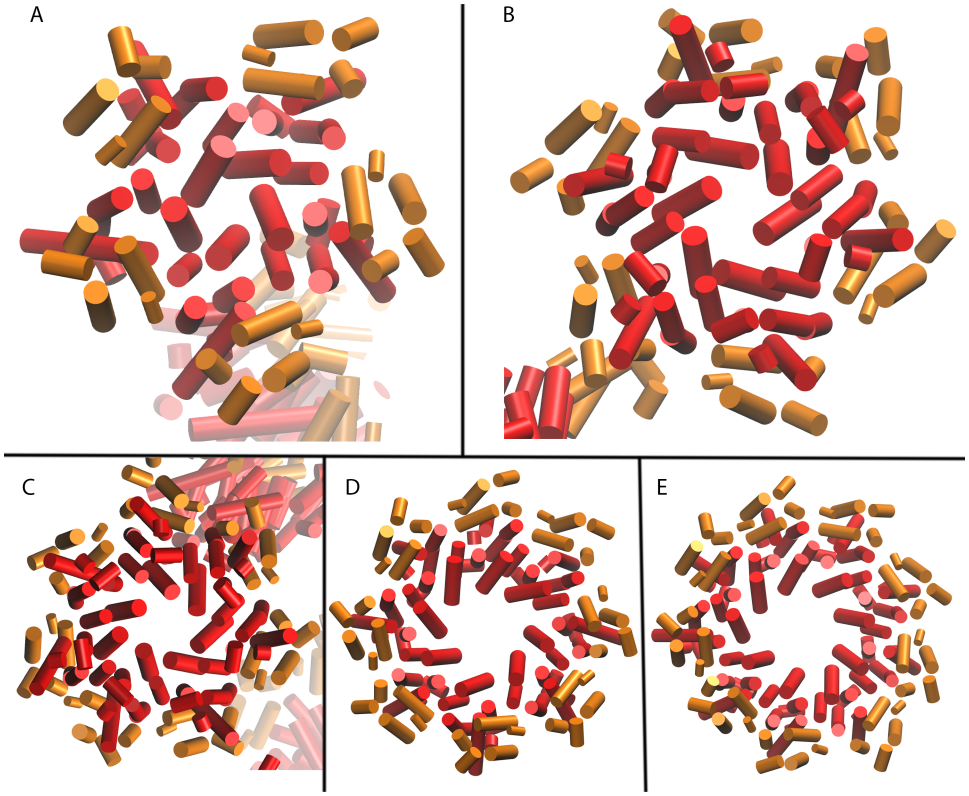

Figure 18: The various oligomers observed can be seen here, including (A) pentamer, (B) hexamer, (C) septamer, (D) octamer, and (E) an oligomer comprised of 9 monomers.

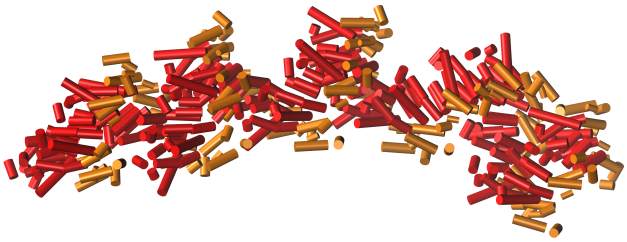

Figure 19: Spirals that were observed in the late stages of runs with not dimeric or trimeric interactions and other interfaces set to 8.0 kcal/mol.

SIGNIFICANT RUNS

System 84

Parameters

System 84 was a system of 105 dimers run on a local machine using a predefined input system. The run was performed on unoptimized code using the GNU Fortran compiler. The input system was derived from part of a lattice consisting of monomers and dimers. Due to the adaptation of the system to fit the pure dimeric format of the simulation, some collisions were evident amongst the subunits. In order to correct for this, the subunits were initially shifted randomly to free the system of these collisions. Any subunits that remained a problem were either manually moved to an acceptable location or removed from the system.

Process

After the simulation was started, the subunits initially dissociated from the input system into a seemingly random distribution. After this, the subunits began reforming a lattice. The simulation was allowed to run for several weeks. The simulation had to be stopped and picked up several times due to local system reboots. After about a week of wall time, the simulation produced formations resembling hexameric lattices.

Results

System 84 was one of the earliest systems to show definite signs of hexameric lattice assembly. Not only did it produce a system of connected hexamers, but at one point on the lattice a double layered hexamer is visible(Figure 20). At some points during the simulation, it did appear that curvature began to form, but for the most part the assembly still appeared flat. It is believed that the smaller system size allowed for a faster assembly than earlier, larger systems.

Curved Assembly

Parameters

This simulation was run on one of the earliest versions of the OpenMP optimized code listed as sections_7-16 under the optimizations section. It was run on a dedicated networked machine. The code was compiled using the GNU Fortran compiler using the `-O3` optimization flag and OpenMP enabled. The system contained 105 dimers and was initialized to a spacing of 10 nm.

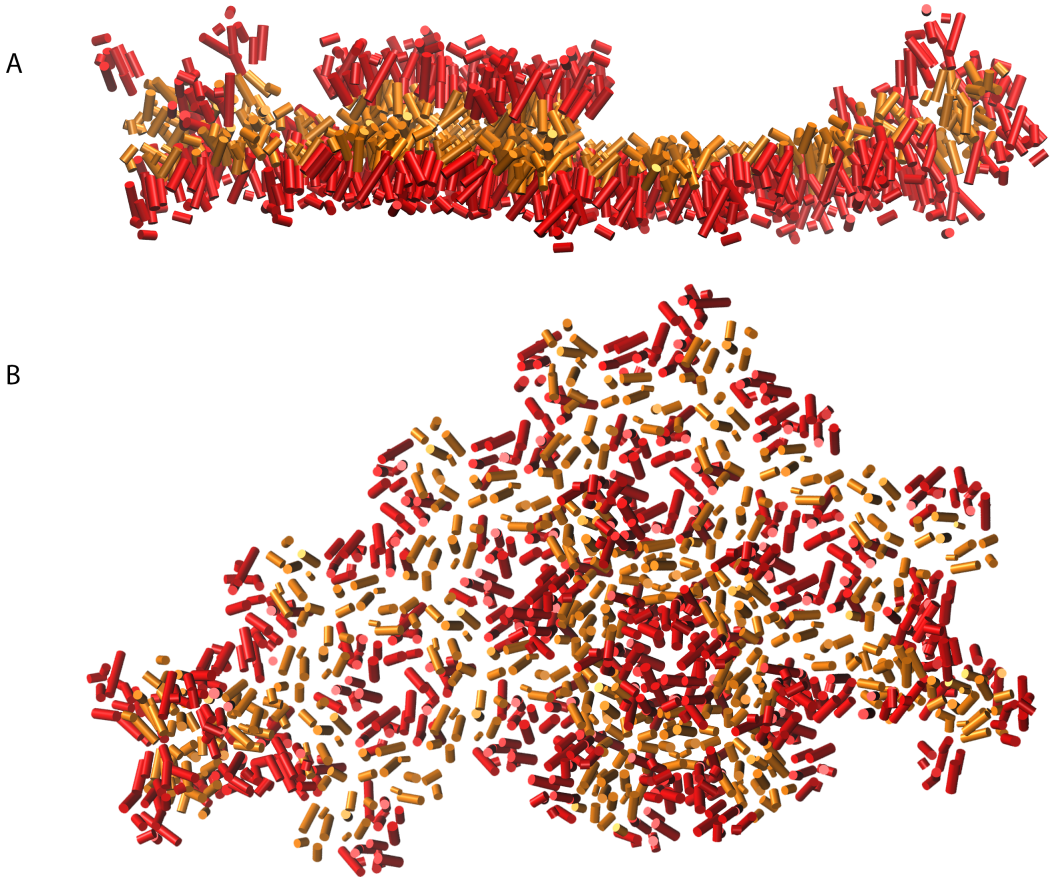

Figure 20: A single, coherent hexameric lattice is visible in the assembly. The double-layered portion of the lattice is apparent in the side view(top). The hexameric lattice is visible in the top-down view(bottom).

Process

The simulation began by annealing the system, and began as a randomized collection of subunits. After a few days of wall time, the subunits managed to successfully form a curved lattice. After a significant amount of wall time, the lattice again appeared flat, similar to earlier runs.

Results

This system was one of the first where the curvature of the simulated lattice corresponded to *in-vitro* observations(Figure 21). Radii between hexamers in the assembly averaged to about 53 nm. Radii between hexamers and other nearby oligomers averaged to about 45 nm. It is unclear what factors lead to this assembly, since other runs of identical code did not yield a curved assembly. There is also the flattening of the lattice after the curvature was observed to consider. Questions remain as to exactly how the curved lattice was formed and why it eventually flattened. According to Zhao, et al. the diameter of *in vitro* tubes ranged from about 40 nm to 50 nm, which is less than what was observed[17].

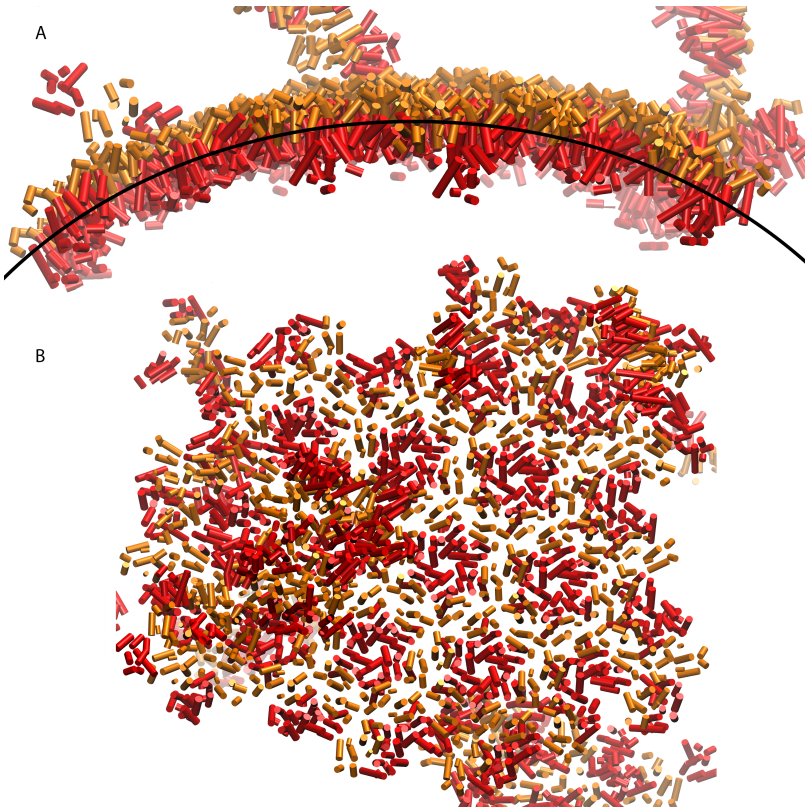

Figure 21: The curvature can be seen traced in the side profile (top). The hexameric lattice can be seen from the top-down view(bottom).

CONCLUSION

The results put forward in this thesis show this simulations ability to capture broad CA structures through the use of a coarse grained model. The simulation was capable of producing hexameric lattices of variable curvature under sufficient conditions. Conclusions can be drawn from this simulation regarding the assembly of the HIV-1 capsid.

It can be seen that a well formed dimeric interface is essential for the proper assembly of the capsid. It also appears that the existence of the protein as a dimer in solution is responsible for the prevalence of hexamers. When tested with monomeric subunits, other oligomers, such as the pentamer, were observed.

The strength of the trimeric interface also has profound effects on the assembly. The trimeric interface was found to be responsible for the appearance of double-layered assemblies. When the trimeric interface was weakened, double-layered assemblies were less common, if present at all. If the trimeric interface strength is taken to be greater than the other interfaces, the assembly will fail completely. In a study by Chen and Tycko, a strong trimeric interface did not seem to impede assembly, but the subunits were confined to a two-dimensional plane[28]. The problem seems to manifest when the system is extrapolated into three dimensions due to the increased freedom of the subunits.

Further Study

The course grained model and simulation used in this study can be easily modified to test a variety of parameters to see how they affect the assembly of the HIV-1 capsid. It may also be possible to modify the code such that it takes into account the flexibility between the NTD and CTD structures within monomeric subunits, and to see how this may affect the assembly. It may also be possible to modify the manner in which different interfaces interact to improve the accuracy

of the simulation. On a much grander scale, the simulation could be reworked to run through true dynamics rather than an MC approach.

This study also leaves some questions unanswered. What causes the curvature of the HIV-1 capsid? How are oligomers arranged in the authentic HIV-1 capsid? These questions may be answered by the modification of parameters or through changes to the code.

APPENDIX

Summary of Scripts

In simulating and deriving data, many Fortran and TCL scripts were used. A brief overview of some of the scripts that were used in obtaining data during this study and their intended functions is given here.

Fortran

Assemble_cubeC

This Fortran program contains the main simulation that was used for the runs consisting of dimeric subunits in this study. From an input file, it determines the initial spacing and temperature, and whether or not to resume a previous run or to begin a new one.

Assemble_monomer

Similar to the Assemble_cube code, except this simulation consists of monomeric subunits.

curvature

This scripts was designed to analyze the number of hexamers and the curvature between adjacent hexamers in a system using geometric conditions. The user will be prompted to input an arbitrary starting step and the number of files. It can convert a three-point output document into the full system, and outputs hexamer data to "hex.dat" and curvature data to "hex_rad.dat."

draw_system4movie subunit_num step file

This script was used to generate most of the images from simulations with dimeric subunits. It requires the user to input the number of subunits in the simulation, the desired step from the input file, and the file from which to take the data. If no file name is input, it is assumed to be "sys_bc.dat." It will output a full system into VMD.

3mon_draw subunit_num step file

This script was used to generate most of the images from simulations with monomeric subunits. It works in the same way as draw_system4movie.

LIST OF REFERENCES

[1] CDC. 2011. Diagnoses of HIV infection and AIDS in the United States and dependent areas, 2011. *HIV Surv. Rep.*. 23.

[2] CDC. 2012. Monitoring selected national HIV prevention and care objectives by using HIV surveillance data- United States and 6 U.S. dependent areas- 2010. *HIV Surv. Supp. Rep.*. 17(3):part A.

[3] T. Douglas, M. Young. 2011. Viruses: making friends with old foes. *Science*. 312:873-875.

[4] Retrovirus. 2013. In Encyclopædia Britannica. Retrieved from http://www.britannica.com

[5] E. Urano, N. Kuramochi, R. Ichikawa, S. Y. Murayama, K. Miyauchi, H. Tomoda, Y. Takebe, M. Nermuit, J. Kumano, Y. Morikawa. 2011. Novel postentry inhibitor of human immunodeficiency virus type 1 replication screened by yeast membrane-associated two-hybrid system. *Antimicrob. Agents Chemother.*. 55:4251-4260.

[6] A. J. Price, A. J. Fletcher, T. Schaller, T. Elliot, K. Lee, V. N. KewalRamani, J. W. Chin, G. J. Towers, L. C. James. 2012. CPSF6 defines a conserved capsid interface that modulates HIV-1 replication. *PLoS Pathog.*. 8(8):e1002896.

[7] J. Shi, J. Zhou, V. B. Shah, C. Aiken, K. Whitby. 2011. Small-molecule inhibition of human immunodeficiency virus type 1 infection by virus capsid destabilization. *J. Virol.*. 85:542-549.

[8] J. Jiang, S. D. Ablan, S. Derebail, K. Hercik, F. Soheilian, J. A. Thomas, S. Tang, I. Hewlett, K. Nagashima, R. J. Gorelick, E. O. Freed, J. G. Levin. 2011. The interdomain linker region of HIV-1 capsid protein is a critical determinant of proper core assembly. *Virology*. 421:253-265.

[9] K. A. Matreyek, S. S. Yücel, X. Li, A. Engelman. 2013. Nucleoporin NUP153 phenylalanine-glycine motifs engage a common binding pocket within the HIV-1 capsid protein to mediate lentiviral infectivity. *PLoS Pathog.*. 9(10):e1003693.

[10] E. L. Yufenyuy, C. Aiken. 2013. The NTD-CTD intersubunit interface plays a critical role in assembly and stabilization of the HIV-1 capsid. *Retrovirology*. 10:29.

[11] C. Soto, B. R. Ratna 2010. Virus hybrids as nanomaterials for biotechnology. *Curr. Op. Biotech.*. 21:426-438.

[12] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller 1953. Equation of state calculations by fast computing machines *J. Chem. Phys.*. 21(6):1087-1092.

[13] B. K. Ganser-Pornillos, M. Yeager, W. I. Sundquist. 2008. Structural biology of HIV assembly. *Curr. Op. Struct. Biol.*. 18:203-217.

[14] J. A. G. Briggs, T. Wilk, R. Welker, H. Kräusslich, S. D. Fuller. 2003. Structural organization of authentic, mature HIV-1 virions and cores. *EMBO J.*. 22(7):1707-1715.

[15] A. Zlotnick. 2003. Are weak protein-protein interactions the general rule in capsid assembly? *Virology*. 315:269-274.

[16] M. Yeager. 2011. Design of *in vitro* symmetric complexes and analysis by hybrid methods reveal mechanisms of HIV capsid assembly. *J. Mol. Biol.*. 410:534-552.

[17] G. Zhao, J. Perilla, E. Yufenyuy, X. Meng, B. Chen, J. Ning, J. Ahn, A. M. Gronenborn, K. Schulten, C. Aiken, P. Zhang. 2013. Mature HIV-1 capsid structure by cryo-electron microscopy and all-atom molecular dynamics. *Nature*. 497:643-646.

[18] B. K. Ganser, S. Li, V. Y. Klishko, J. T. Finch, W. I. Sundquist. 1999. Assembly and analysis of conical models for the HIV-1 core. *Science*. 283:80-83.

[19] L. Deshmukh, C. D. Schwieters, A. Grishaev, R. Ghirlando, J. L. Baber, G. M. Clore. 2013. Structure and dynamics of full length HIV-1 capsid protein in solution. *J. Am. Chem. Soc.*.

[20] O. Pornillos, B. K. Ganser-Pornillos, M. Yeager 2011. Atomic-level modelling of the HIV capsid. *Nature*. 469:424-427.

[21] V. Krishna, G. S. Ayton, G. A. Voth. 2010. Role of protein interactions in defining HIV-1 viral capsid shape and stability: a coarse-grained analysis. *Biophys. J.*. 98:18-26.

[22] R. K. Gitti, B. M. Lee, J. Walker, M. F. Summers, S. Yoo, W. I. Sundquist 1996. Structure of the amino-terminal core domain of the HIV-1 capsid protein. *Science*. 273:231-235.

[23] C. Momany, L. C. Kovari, A. J. Prongay, W. Keller, R. K. Gitti, B. M. Lee, A. E. Gorbalenya, L. Tong, J. Mclure, L. S. Ehrlich, M. F. Summers, C. Carter, M. G. Rossman. 1996. Crystal structure of dimeric HIV-1 capsid protein. *Nat. Struct. Biol.*. 3:763-770.

[24] T. R. Gamble, S. Yoo, F. F. Vajdos, U. K. von Schwedler, D. K. Worthylake, H. Wang, J. P. McCutcheon, W. I. Sundquist, C. P. Hill 1997. Structure of the carboxyl-terminal dimerization domain of the HIV-1 capsid protein. *Science*. 278:849-853.

[25] J. M. A. Grime, G. A. Voth. 2012. Early stages of the HIV-1 capsid protein lattice formation. *Biophys. J.*. 103:1774-1783.

[26] I. L. Byeon, X. Meng, J. Jung, G. Zhao, R. Yang, J. Ahn, J. Shi, J. Concel, C. Aiken, P. Zhang, A. M. Gronenbom. 2009. Structural convergence between Cryo-EM and NMR reveals intersubunit interactions critical for HIV-1 capsid function. *Cell*. 139:780-790.

[27] M. F. Hagan. 2013. Modeling viral capsid assembly. Manuscript submitted for publication.

[28] B. Chen and R. Tycko. 2011. Simulated self-assembly of the HIV-1 capsid: protein shape and native contacts are sufficient for two-dimensional lattice formation. *Biophys. J.*. 100:3035-3044.

[29] B. Chen. 2013. Accurate and efficient algorithm for fast collision detection and distance measurement between cylinders. Unpublished manuscript, University of Central Florida, Orlando, FL.

[30] P. Ceres and A. Zlotnick. 2002. Weak Protein-Protein Interactions Are Sufficient To Drive Assembly of Hepatitis B Virus Capsids. *Biochem.*. 41:11525-11531