Phases of polymer systems in solution studied via molecular dynamics

by

Joshua Allen Anderson

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Condensed Matter Physics

Program of Study Committee:
Alex Travesset, Major Professor
Steve Kawaler
Surya K. Mallapragada
Joerg Schmalian
David Vaknin

Iowa State University

Ames, Iowa

2009

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1. INTRODUCTION

1.1 General introduction

Polymers are amazingly versatile molecules with a tremendous range of applications. Our lives would be very different without them. There would be no multitudes of plastic encased electronic gizmos, no latex paint on the walls and no rubber tires, just to name a few of the many commonplace polymer materials. In fact, life as we know it wouldn't exist without polymers as two of the most essential types of molecules central to cellular life, Proteins and DNA, are both polymers! [1] With their wide range of application to a variety of uses, polymers are still a very active field in basic research. Of particular current interest is the idea of combining polymers with inorganic particles to form novel composite materials. [2]

As computers are becoming faster, they are becoming all the more powerful tools for modeling and simulating real systems. With recent advances in computing on graphics processing units (GPUs) [3–7], questions can now be answered via simulation that could not even be asked before. This thesis focuses on the use of computer simulations to model novel polymer-inorganic composite systems in order to predict what possible phases can form and under what conditions. The goal is to provide some direction for future experiments and to gain a deeper understanding of the fundamental physics involved. Along the way, there are some interesting and essential side-tracks in the areas of equilibrating complicated phases and accelerating the available computer power with GPU computing, both of which are necessary steps to enable the study of polymer nanocomposites.

1.2 Overview and thesis organization

Following this introduction, chapter 2 provides a very brief overview of polymer physics focused on those aspects important to the research performed in this thesis. Then, chapter 3 introduces molecular dynamics as a simulation technique and its application to polymer systems.

After these introductory chapters, there are a number of published papers included in the thesis. The first is in chapter 4. It calculates the phase diagram of a particular pentablock copolymer using molecular dynamics (MD). As first author, I performed all of the simulations, all of the data analysis, and wrote the bulk of the paper discussing the results.

This initial study is motivated by experiments performed by Prof. Mallapragada's group [8–10] which are aimed at developing polymer-inorganic composite materials. Studying such complicated systems with MD first requires a good understanding of the simpler case of just multiblock polymers in solution, a problem that was still poorly understood at the time [11]. In order to gain this understanding for the cubic phases relevant to the experiments, another study is undertaken and another paper published, included in chapter 5. As with the previous paper, I performed all of the simulations and data analysis and wrote the bulk of the paper.

A *significant* amount of computer time is needed in order to solve the problems associated with equilibration in this second paper. At the time of its completion, general purpose computing on the GPU was starting to gain a significant amount of attention. And with NVIDIA's release of CUDA, the prospects for running scientific applications on the GPU became much more promising. Using it, some astronomy simulations developed at the time reported greater than a factor of 100 speedups for certain calculations [4,5]. At this level, a single GPU has the awesome potential to replace a computer cluster at a fraction of the cost.

Thus we began a fun "side project" over the summer into the feasibility of using GPUs to accelerate molecular dynamics. The result, after only a few months of work, is a fully functional code with significant speedups. All the details are published in the paper included in chapter 6. We are the *first* to publish all the algorithms needed to efficiently implement MD fully on the GPU. In this paper, all of the algorithmic design and programming expertise

come from me and I wrote the entire paper.

Finally, with an understanding of equilibrating phases in MD and the power of GPU computing to perform the massive amount of simulations needed, a final paper (for the purposes of this thesis, at least) is published detailing an extensive study of the phases of polymer-inorganic composite materials included in chapter 7. As with the other papers of this type, I perform all the simulations, most of the analysis, and write the bulk of the paper.

Of course, this final extensive paper would not have been possible without the additional preliminary work in ref. [12] by Chris Knorowski, a REU student I mentored for a summer, and the subsequent work by Rastko Snepnek in ref. [13], whom I collaborated with.

As an aside, but an important one, chapter 8 looks at numerical errors in molecular dynamics calculations. The focus is on the differences in GPU computing vs. CPU computing in regard to these errors and the analysis serves as a first step in bringing the two in line.

The final chapter, chapter 9, wraps up with some conclusions.

CHAPTER 2. INTRODUCTION TO POLYMERS

2.1 Polymer fundamentals

2.1.1 Monomers

Polymers are large molecules made up of many repeating units called *monomers*. Chemically, monomers are often variations on hydrocarbons with the covalent bonding between carbon atoms providing the links between monomers. All hydrocarbons (except methane) are, of course, polymers given this consideration. section 2.1 shows polyethylene as an example. There is a zoo of different monomers available to build polymers from, each with unique properties useful in different applications. [1, 14]

Discussing particulars of the chemistry is beyond the scope of this thesis. Instead, polymers are presented throughout as an abstracted bead-spring model. Each monomer is represented by a bead with a position and mass. The bonds that string these beads into polymers are represented as springs, as shown in section 2.1.

2.1.2 Types of polymers

The simplest and most common type of polymer is one where all monomers are connected to form a long linear chain of n monomers. Other configurations are possible with a few examples shown in section 2.1. For instance, at certain points along the main chain, shorter chains can branch off. Or long chains could all be attached to a central monomer, forming a star pattern. [14] Of course, the possible permutations do not stop here. Branches can be linked to other branches, which are linked to other branches all the way up in a very complicated topology. While it may seem like a simple thought experiment, all these possible

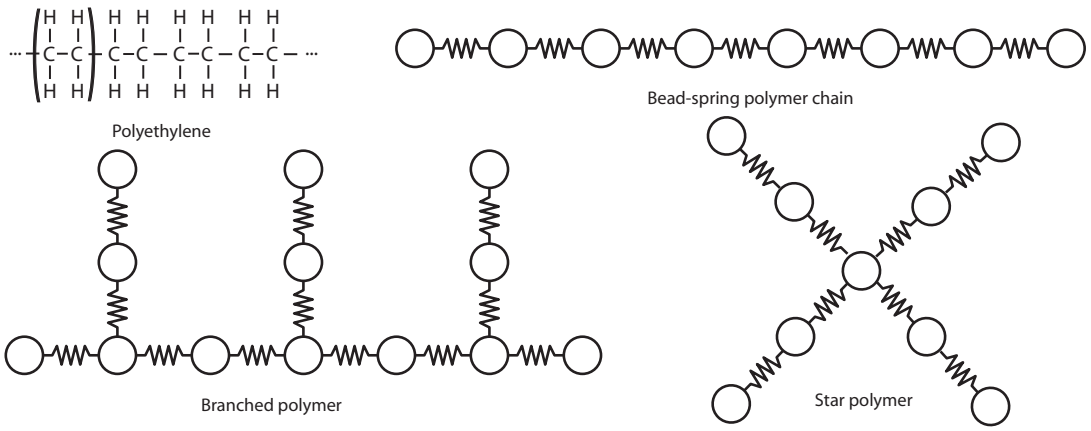

Figure 2.1 Examples of different types of polymers

branching patterns are not just the subject of theoretical interest. Chemists have long been synthesizing polymers with controllable architectures, and some applications depend on the unique properties provided by the branching. [1, 14]

2.1.3 Polydispersity

One important issue worth mention on the subject of synthesizing polymers is that of *polydispersity*. Polymerization is a chemical reaction occurring in a soup of monomers and partially formed polymers. Random thermal effects may result in the final polymers not all ending up with exactly the same length, or some small amount of branching may have occurred where linear chains were desired. [14] The degree to which not all of the polymers in a system are the same is called the polydispersity, and limiting it is an important challenge to experimentalists. Of course, dealing with the theory of polymers in this thesis, *mono*disperse systems, where all polymers are identical, can easily be created and studied. Polydispersity can of course be incorporated into models, too, when it is necessary.

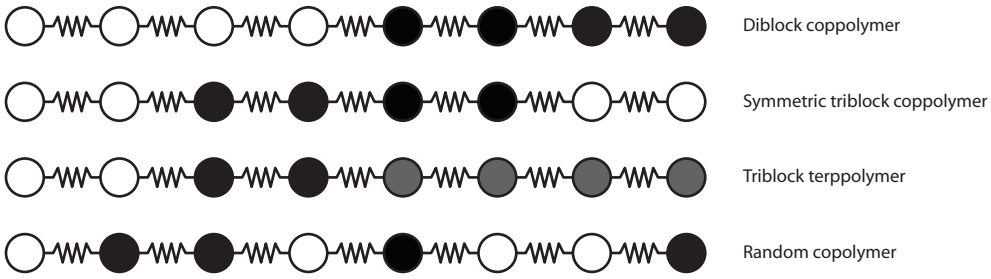

Figure 2.2 Block copolymer examples

2.1.4 Block polymers

Just as there is virtually no limit on the branching architecture of polymers, there is no limit forcing all monomers to be of the same type. *Heteropolymers* can be created with two or more different types of monomers spread randomly across the chain or organized into blocks as shown in subsection 2.1.4. [1] Block copolymers, polymers with two different types of monomers grouped in blocks, have particularly interesting properties discussed later in section 2.2 and are the main subject in the research papers included in this thesis (chapters 4, chapter 5, and 7).

2.1.5 Polymers in solution

A system of many polymers together with nothing else present is called a polymer *melt*. The physics is relatively well understood for such systems of polymers with simple architectures, like homopolymers and diblocks [1, 14]. The details are skipped over here as the research area this thesis focuses on involves systems of polymers in *solution*. A solution is a liquid of solvent molecules into which some material, polymers in this case, has been dissolved. Water and ethanol are two household examples of solvents. Many industrial solvents, such as cyclohexane, are also often used in conjunction with polymers.

Working up to the level of studying many interacting polymers first requires an understanding of the conformations of a *single* polymer chain in solution. The picture to have in

mind for the following discussion is that of a long chain of monomers surrounded by a sea of solvent. At first glance, it would seem that the flexibility or stiffness of the individual bonds to bending should play a big role in determining the possible conformations of the polymer. Indeed, locally, a trio of Carbon atoms in the chain $C-C-C$ are quite stiff to bending. However, over a long enough length scale many small bends can add up to a large one [1, 14], and at this level the polymer appears completely flexible. The length scale this occurs at is called the *persistence length* [14] or *Kuhn length* [1]. Its value depends on the temperature and the chemical makeup of the monomer [14]. No matter what this length is, in the idealized case of an infinitely long polymer chain we can always "zoom out" to a length scale much larger than the Kuhn length and observe an apparently completely flexible polymer.

What does have an impact on the conformation of the polymer is the type of interactions between the monomer and the solvent. Special terms apply when that solvent is water. *Hydrophilic* monomers like to be in contact with water. A polymer chain of hydrophilic monomers will spread out as much as it can in what is called a *coil*. Hydro*phobic* monomers prefer each other more than they prefer water. A polymer chain of these monomers will collapse into a *globule*, shielding as many monomers as can be from contact with water. This concept can be generalized to other types of monomers in solvents other than water. A polymer in a *good solvent* spreads into a coil configuration, while it collapses into a globule in a *poor solvent*. There is a halfway point between the two called a *theta solvent*, which is best explained in the more quantitative framework in subsection 2.1.6. [1]

2.1.6 Scaling

Long polymer chains exhibit a fractal scaling behavior, demonstrated in subsection 2.1.6. The effects of this behavior are such that if portions of the same polymer are examined over different length scales, the conformations have *identical* statistical properties in properly scaled coordinates. [1] This feature of polymers is very useful in efficiently modeling them on the computer, the reasons for which are described in subsection 3.5.1.

Scaling behavior can also be applied to analytical calculations of polymer properties. The

Figure 2.3 One of these two panels shows only a portion of a polymer generated from a random walk. The other shows the entire polymer, but only half of the monomers are displayed and the view is zoomed out by a factor of two. Can you tell which is which? (Hint: The partial view is the lower right quadrant of the full view). This visual self-similarity demonstrates the fractal scaling nature of polymers.

most famous example of this is the Flory exponent [14]:

$$\langle R_G \rangle = a \cdot n^{\nu} \tag{2.1}$$

where $\langle\rangle$ denotes an ensemble average. R_G is the radius of gyration of the polymer, defined by

$$R_G^2 = \frac{1}{n} \sum_{i=1}^{n} (\vec{r}_i - \vec{r}_{\text{cm}})^2 \tag{2.2}$$

, where $\vec{r}_i$ are the positions of the monomers, $\vec{r}_{\text{cm}}$ is the center of mass position, and n is the number of monomers. As a root mean square of the monomer positions, the radius of gyration is a measure of the size of the space that the polymer occupies. With that in mind, Equation 2.1 says that the size of the polymer scales with the number of monomers n to the power ν, with some constant of proportionality, a.

What is truly remarkable is that ν takes on one of three *discrete* values depending on the solvent. In a good solvent, Flory's theory predicts $\nu = 3/5$ which is extremely close to the best available numerical result of 0.588 [14]. In a theta solvent $\nu = 1/2$ and in a poor solvent,

$\nu = 1/3$. The discrete values ν can take on are well understood from the work of DeGennes [14]. He mapped the problem into a spin model where renormalization group theory dictates the allowed value of the exponents. DeGennes was awarded the 1991 Nobel prize in physics in part for this discovery.

The last case with $\nu = 1/3$ can be easily understood with an intuitive argument. In a poor solvent, the polymer wants to collapse into as small a space as it can in order to shield the hydrophobic beads from the solvent. What better shape is there to accomplish this task than a sphere? From simple geometry, the radius of a sphere scales as the volume to the 1/3 power, thus explaining $\nu = 1/3$ as the volume of a sphere of tightly packed monomers will scale proportionately with the number of monomers in the sphere.

The scaling behavior of polymers can also be taken advantage of to calculate many other properties [14]. Detailed discussions on these analytical calculations are beyond the scope of this thesis and there are excellent textbooks on the subject [1, 14].

2.2 Phases

2.2.1 Segregation

As a first step in predicting the behavior of large systems of many block copolymers, an understanding of the tendency of two different components to mix is needed. The *Flory interaction parameter* χ nicely wraps up all of the microscopic interactions, temperature and pressure effects, etc... [1,14]. The details behind its derivation are not important here. What is important is the the effects of the parameter and how it can be used to model polymer systems numerically.

Given a system defined over a region of space, let $\phi_{\mathrm{A}}(\vec{r})$ be the fraction of the volume occupied by component A and $\phi_{\mathrm{B}}(\vec{r})$ be same for component B. For the purposes of this discussion it does not matter whether one of these components is a monomer in a polymer and the other is solvent, or if they are two different components in a block copolymer. The Flory interaction parameter can be applied in both cases. χ is relevant in a type of mean field approximation where in a given small unit of space, ϕ_{B} is constant and thus $\phi_{\mathrm{A}} = 1 - \phi_{\mathrm{B}}$.

The utility of χ comes with the following definition, which is regular solution theory applied to polymer physics:

$$F_{\text{int}} = \chi \cdot \phi_{\text{A}}(1 - \phi_{\text{A}}) \cdot k_{\text{B}}T \tag{2.3}$$

. It says that the free energy in each small unit of space is given by the product of χ, the densities of the components and the temperature. When χ is positive and large, minimizing the free energy requires minimizing the product $\phi_{\text{A}}(1-\phi_{\text{A}})$ and can be done by making either ϕ_{A} or $1 - \phi_{\text{A}} = \phi_{\text{B}}$ go to 0. Thus large positive values of χ encourage the segregation of the A and B components. Similarly, a negative χ leads to a minimum free energy when the product of densities is maximal. This occurs when the densities are each 0.5 and thus mixing is promoted. [14]

Using this theory as a starting point, a number of properties of polymer systems can be calculated analytically. [1,14] An exposition of such calculations is well beyond the scope of this thesis. The Flory interaction parameter does appear again in section 2.3 in a brief discussion on various numerical modeling strategies.

2.2.2 Self-assembly in solution

Moving on to a more concrete example appropriate to the types of polymers studied in detail later in this thesis, let's look at what happens with *amphiphilic* polymers in solution. Amphiphilic polymers have both hydrophilic and hydrophobic monomers. Specifically, say we have a system of diblock polymers like the one shown in subsection 2.1.4 where the first block is hydrophilic and the second is hydrophobic. Immediately after being placed in solution, the hydrophilic half is happy where it is, but the hydrophobic half is not. In a process called *self-assembly*, the hydrophobic blocks of many polymers find each other and aggregate forming a *micelle*. subsection 2.2.2 is a schematic diagram for a micelle. The hydrophobic blocks are all contained within the dense *core* of the micelle where the solvent has been expelled while the hydrophilic blocks reach out into the solvent in the *corona*, shielding the core from contact with the solvent.

The typical example of a micelle is a spherically shaped object, but they can come in many

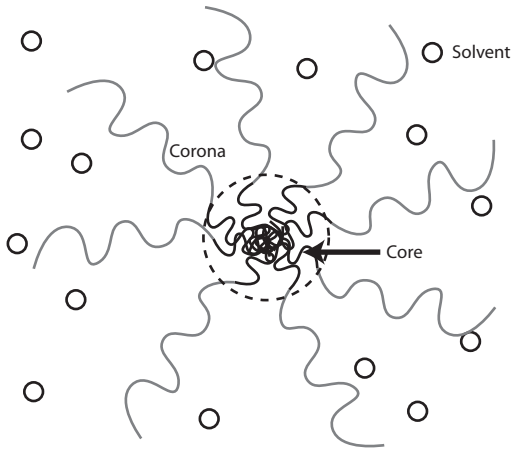

Figure 2.4 The dark lines in the middle are the hydrophobic block of the polymer and aggregate to form the core of the micelle. Gray lines around it are the hydrophilic block of the polymer forming the corona. The presence of the solvent is indicated by small circles.

shapes and sizes. Being made of a conglomeration of flexible polymers, a roughly spherical micelle can be squished and bent in various ways depending on its environment, distorting it from a truly spherical shape. Not all micelles are even roughly spherical. At higher polymer concentrations, cylindrical micelles are common.

In a system consisting of many micelles, the possibility exists that they arrange in patterns that exhibit long range order. For instance, spherical micelles in solution can line up on a cubic lattice (see chapter 5 for a detailed study). Or cylindrical micelles can pack together in ordered hexagonal or square packing patterns (see chapter 7 for many examples). These are just a few examples of the many phases that can result with amphiphilic polymers in solution. More details regarding the possible phases and their structure can be found in the research papers included in this thesis.

2.3 Numerical modeling and simulation methods

The polymer in subsection 2.1.6 was created with a very simple numerical technique called a *random walk*. Starting with a monomer at an initial point, a step in a random direction is

taken and another monomer placed there. Yet another step in another random direction is then taken and a third monomer is placed. This process repeats until the n monomers making up the polymer are all placed. This method is in a class of simulations called *Monte Carlo* simulations because they model physical systems with a series of events which occur with a random probability (with the proper distribution, of course). [1]

Methods exist to apply Monte Carlo techniques to model more complicated polymer systems, including the amphiphilic polymer systems mentioned in the previous section. One way this is done is by representing each of the monomers on the points of a cubic lattice. A set of initial polymers is created with the random walk technique and then perturbed by a series of *moves* that each slightly rearrange the polymers (i.e. by moving a monomer from one lattice point to a neighboring one). A potential energy function is chosen that models the essential physics and is employed as a filter to accept or reject each potential move (sometimes, the potential energy is calculated with Equation 2.3). If a move lowers the energy, it is always accepted. If it instead raises the energy, it is accepted with a probability

$$p_{i\to j} = \exp\left(-\frac{E_j - E_i}{k_{\mathrm{B}}T}\right) \tag{2.4}$$

where E_i and E_j are the energy of the system before and after the move in question. In this way, the ensemble of all states visited by performing many moves is that of the NVT ensemble of statistical mechanics. [1] While not as commonly used in the literature, it is possible to perform similar simulations without the restriction that the monomers sit on a lattice.

In a *self consistent field theory*, an initial density profile $\phi(\vec{r})$ is assumed along with the symmetries of the phase under study. Then, Equation 2.3 together with the free energy of the non-interacting polymer blocks is used to calculate an apparent "force" which is then used to update the density. At this point Equation 2.3 is used again and the process is repeated until an unchanging self-consistent solution is found. [14] By its implicit assumptions, self consistent field theories are particularly appropriate to describe melts.

Molecular dynamics (MD) represents each monomer as a single particle and solves the equations of motion to calculate the behavior of the system over time (more details in chapter 3). This is the method chosen for modeling polymer systems in this thesis. Without going into

too many details on the pros and cons of each method, MD offers some distinct advantages. First, the calculations are simple and adding additional types of particles and interactions is straightforward. In self-consistent field theory, for example, the addition of inorganic particles (see chapter 7 for a relevant research topic) must be done in a very unnatural way by subtracting delta function voids from the density. Second, the calculations in MD can be efficiently executed in parallel on large computer systems, allowing more polymers to be simulated for longer times. Finally, when using MD something can be learned about the actual time dependent dynamics of the system in question, which is not possible with Monte Carlo nor self consistent field theory as they are ensemble approaches.

CHAPTER 3. INTRODUCTION TO MOLECULAR DYNAMICS

3.1 The method

3.1.1 Definition

Molecular dynamics (MD) is a very simple but powerful simulation technique. In a way, it can be considered as a *brute-force* approach to solving statistical mechanics problems. It starts by representing every atom of the system in question as a point in 3-space $\vec{r}_j$, with j as the index assigned to the atom from 1 to N. Some Hamiltonian is given, e.g. a Newtonian $H = K + V(\vec{r}_1, \vec{r}_2, ...\vec{r}_n)$, to specify the physics these atoms obey. MD makes no attempt to finesse a solution given this information. In the simplest terms it is the solution of the equations of motion resulting from the Hamiltonian. [15] Perhaps this doesn't sound like much of a task, as physics students do this every day in classical mechanics class for systems of a few particles. But in a research setting MD is typically applied to anywhere from tens of thousands of particles up to hundreds of millions [6] and limited only by the computer power that can be brought to bear.

3.1.2 Numerical integration

Tackling the solution of thousands of coupled differential equations of motion is the task of the numerical integration routine. Such methods are usually derived by taking the Taylor series expansion of the variable in question. Depending on the way that terms are combined and cancelled, many different methods are available. [16] A very simple and often used scheme for numerically integrating Newtonian motion is called Verlet integration. It can be derived

simply by first taking the forward Taylor series expansion of $\vec{r}_j(t)$,

$$\vec{r}_j(t+\delta t) = \vec{r}_j(t) + \vec{v}_j(t)\cdot\delta t + 1/2\cdot\vec{a}_j(t)\cdot\delta t^2 + 1/6\cdot\vec{b}_j(t)\cdot\delta t^3 + O(\delta t^4) \tag{3.1}$$

(here the first second and third derivatives have been replaced by velocity $\vec{v}$, acceleration $\vec{a}$ and jerk $\vec{b}$) and then taking the same expansion but stepping *back* in by time δt

$$\vec{r}_j(t-\delta t) = \vec{r}_j(t) - \vec{v}_j(t)\cdot\delta t + 1/2\cdot\vec{a}_j(t)\cdot\delta t^2 - 1/6\cdot\vec{b}_j(t)\cdot\delta t^3 + O(\delta t^4) \tag{3.2}$$

. Add those two equations together and the δt and δt^3 terms cancel out leaving an expression involving only the position at various times and the acceleration:

$$\vec{r}_j(t+\delta t) = 2\vec{r}_j(t) - \vec{r}_j(t-\delta t) + \vec{a}_j(t)\cdot\delta t^2 + O(\delta t^4) \tag{3.3}$$

.

Reading Equation 3.3 from left to right, it says that the position of a particle j at time $t+\delta t$ can be easily (and approximately) calculated knowing only the two previous saved positions and the current acceleration. The step size δt must clearly be chosen small so that the error term $O(\delta t^4)$ remains negligible (see chapter 8 for more information on such errors). With a small δt, whatever behavior of the modeled system is of interest most likely occurs over the time span of thousands or even millions of time steps. Thus, the core of any MD simulation is performing step after step, each one advancing the simulation slowly forward in time.

Equation 3.3 works well, but is not the most convenient. Often, for purposes of the evaluation of the kinetic energy or other reasons, it is desirable to know both the position and velocity. There is another way to formulate the Verlet integration routine making this possible [15]. It calculates identical results to the one derived here, but tracks particle position and velocity instead of the position at two separate times.

3.1.3 Statistical ensembles

Newtonian mechanics makes for simple examples and descriptions in a thesis, but any real system of interest is probably not conveniently described in the resulting NVE ensemble (constant number of particles N, constant volume, and constant energy). Of course, once

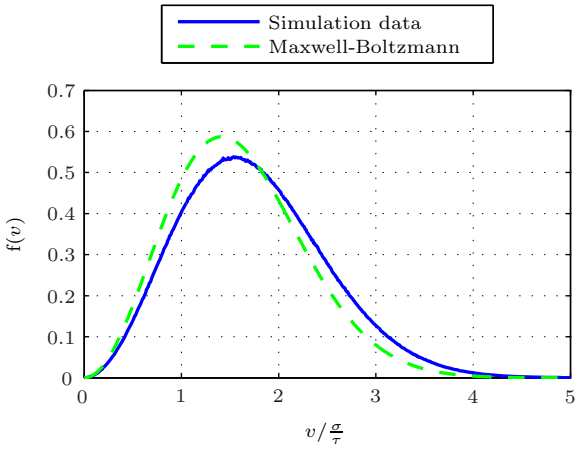

Figure 3.1 Velocity distribution both from simulation in an NVT ensemble and from theory. The quantity $\mathrm{f}(v)\mathrm{d}v$ is the probability of finding a particle with velocity between v and $v + \mathrm{d}v$.

equilibrium is attained the NVE ensemble is identical to any other. But reaching the desired equilibrium by guessing an initial condition and running NVE for a while is nearly impossible. Most real world systems are either held at constant temperature and volume (NVT) or constant pressure and temperature (NPT). Successful methods for modeling these systems in MD have been developed. [17, 18].

Going through the entire derivation of these methods here is beyond the scope of this thesis. As a brief summary, they work by introducing a fictitious degree of freedom into the Hamiltonian H. In the case of NVT integration, the temperature of the system T appears in such a way that when the system temperature T is below the set point, energy is transfered from the fictitious degree of freedom into the system. When the actual temperature T is above the set point, energy is transferred from the system to the fictitious degree of freedom. In this way, the temperature is regulated to always be near the set point and so it is called a *thermostat*. It can be proven that with the particular functional form used, the region in phase space the system explores is exactly the canonical ensemble of statistical mechanics. [17] This property can be demonstrated by comparing the distribution of particle velocities found in a simulation with the theoretical Maxwell-Boltzmann result. Such a comparison is made in

subsection 3.1.3 with good agreement. The agreement could be a little better, however. It is thought that the particular numerical integrator chosen to implement NVT integration in this case is not particularly precise. An investigation into other integrators will be performed in the future, especially in view of the results obtained in chapter 8.

3.1.4 Periodic systems

Computers are finite machines and can only store and handle so many atoms in a given simulation. For instance, there is no possible way every atom in an entire beaker of water could be simulated. Today's computers have enough power and storage for only a tiny, nanometers wide, portion of the liquid. However, accurate results can still be obtained, particularly if periodic boundary conditions are employed correctly. In the simplest case, all atoms are contained in a cubic simulation box and the simulation is programmed so that when an atom leaves one side, it comes in the opposite side. Additionally, atoms near the edge can "see" the atoms on the other side. As long as there are enough atoms in the simulation to obtain good statistics and the size of the box is larger than the length scale of the property under study, there are usually no serious problems. The same technique works for the study of materials with periodic repetitions, i.e. crystalline solids, though depending on the symmetry the crystal might not fit perfectly in a cubic box (see chapter 5 for an example).

3.2 Properties calculated

Fundamentally, MD provides a full description of the position and velocity of every atom at every point in time during the simulation. From this information any of a multitude of system properties can be calculated. Starting with a few basic ones, the kinetic energy of the system can be calculated using the standard equation

$$K = \sum_{j=1}^{N} \frac{1}{2} m_j \cdot v_j^2 \tag{3.4}$$

. From this, the temperature can be calculated using the equipartition theorem

$$T = \frac{2\langle K \rangle}{N_{\mathrm{dof}} k_{\mathrm{B}}} \tag{3.5}$$

(where N_{dof} is the number of degrees of freedom in the system). Many other thermodynamic state variables can be similarly calculated.

Most often in the work done in this thesis, the primary result of interest is the long range order in the phase that has formed over the course of the simulation. Typically, this can be easily identified by eye just by displaying the spatial coordinates $\vec{r}_j(t)$ interactively in 3D rendering software such as VMD [19]. Snapshots saved every so often can even be loaded in and played as a movie, which is often useful when trying to gain an intuitive grasp on the dynamic behavior of the systems.

But the limit of properties that can be calculated certainly does not stop here. Just from the position data alone, quantities such as the pair correlation function can be calculated by averaging over all particles and many configurations. [15] Densities can be averaged and plots made of that. Gyration radii of polymers can be calculated or micelles identified. There are way too many such properties to even list here. For more polymer related examples examples of useful dynamical and structural properties that can be extracted from MD simulation results, see the *observables* sections in the various papers included in this thesis, chapters 4, 5, and 7.

3.3 Potential energy specification

So far, no specific mention has been made of the specification of the potential energy $V(\vec{r}_1, \vec{r}_2, ...\vec{r}_n)$ used to obtain the particle accelerations for use in the numerical integration. As a technique, molecular dynamics is extremely general and can be applied to a wide range of physical systems. The potential energy V is simply the input into the simulation that determines the dynamics to be simulated. It can be calculated from physics as detailed as *ab initio* quantum calculations [20], or from potentials tabulated between atoms and tweaked by fitting techniques to provide accurate results [21], or approximated further to idealized potentials between coarse-grained beads representing groups of atoms, all the way up to potentials drawn by hand to any shape that can be imagined.

Each level of further approximation reduces the complexity of calculation, allowing larger systems to be simulated for more time steps while using the same amount of computer time.

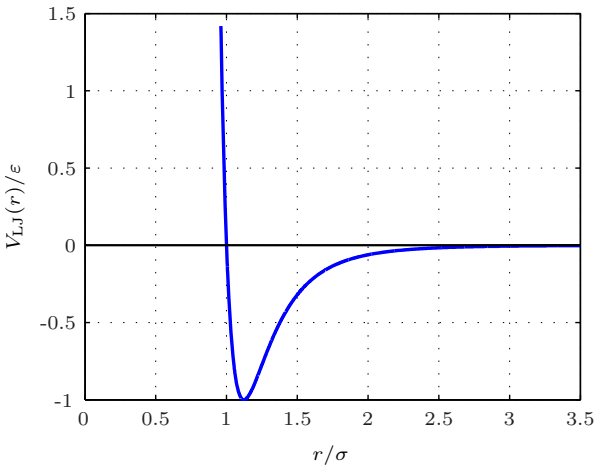

Figure 3.2 The Lennard-Jones pair potential

For example, the state of the art in *ab initio* MD may only be able to simulate a few hundred atoms for total simulation times of mere nanoseconds on the world's fastest supercomputers. In contrast, the coarse-grained polymer simulations in this thesis *effectively* contain hundreds of millions of atoms and run for hundreds of microseconds of simulation time. Of course, each method has its own respective area of application and the level of physics employed in the calculation of V must be chosen appropriate to the problem. More discussion on the level of approximation relevant to polymers is present in subsection 3.5.1.

To make this general discussion more concrete, for the purposes of this thesis (and even a huge majority of MD simulations performed in the community), V is approximated by breaking it into pieces.

$$V(\vec{r}_1, \vec{r}_2, ...\vec{r}_n) \approx \underbrace{\sum_{i=1}^{N} \sum_{j=i+1}^{N} V_{\mathrm{pair}}(\vec{r}_j - \vec{r}_i, i, j)}_{\text{pair energy}} + \underbrace{\sum_{(i,j)\in \text{bonds}} V_{\mathrm{bond}}(\vec{r}_j - \vec{r}_i)}_{\text{bond energy}} \tag{3.6}$$

. First, the pair energy is summed over all pairs of particles. In general, it is expressed as a function of the vector between the particles in the pair but it is most often simplified to a more convenient form in a coordinate system where $\vec{r}_i$ is at the origin and the potential is isotropic:

$$V_{\mathrm{pair}}(\vec{r}_j - \vec{r}_i, i, j) = V(r, i, j) \tag{3.7}$$

. This form is so often discussed that the "pair" subscript is usually dropped for brevity. The indices i and j remain as parameters to the function as the two particles may be of different types and the potential must be allowed to account for this. The *pair potential* $V(r, i, j)$ can take many forms, but the canonical example is the Lennard-Jones potential

$$V_{\mathrm{LJ}}(r, i, j) = 4\varepsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r}\right)^{12} - \left(\frac{\sigma_{ij}}{r}\right)^{6}\right] \tag{3.8}$$

where ε_{ij} is minimum of the potential located at $r = 2^{1/6}\sigma_{ij} \approx \sigma_{ij}$ (see section 3.3). The main features of the Lennard-Jones potential include a hard core repulsion for small r when the atoms are near each other along with a medium-range attraction. It is actually a very accurate potential for modeling interactions between atoms in a noble gas [15], but it is also commonly used in coarse-grained systems.

The bond energy can be simplified in a similar assumed coordinate system. And the simplest model for two bonded particles is, of course, harmonic

$$V_{\mathrm{harm}}(r) = \frac{1}{2}k\left(r - r_0\right)^2 \tag{3.9}$$

where k is the stiffness constant and r_0 is the equilibrium bond length.

In simulations of atomic systems, biomolecular simulations in particular, additional bond terms between 3 and 4 particles are included. [21] They are not used for the polymer simulations in this thesis, so they are not presented in detail here.

3.4 Problem size and calculation efficiency

3.4.1 Scaling and efficiency

Evaluating Equation 3.6 (or more accurately, its derivatives to get the force) is the most time consuming step in performing a MD simulation. Implementing the calculation as a direct interpretation of the forumla results in $N_{\mathrm{bonds}} + N(N-1) \in \Theta(N^2)$ calculations to be performed in each time step. In this case, doubling the size of the simulation would mean that it takes *four* times as long to execute. With N desired in the tens of thousands, the number of calculations to be performed is extremely high.

An approximation is often made to bring the calculations down to a more attainable level. The pair potentials (and forces) are cut off to 0 at $r = r_{\text{cut}}$, which is 3.0σ in all work presented in this thesis. Visual inspection of the Lennard-Jones potential in section 3.3 shows that the potential is near zero at this point, so performing the cut only minimally changes the dynamics. The upshot is that the number of calculations to perform is reduced by a huge factor. Even better, with the right algorithms the time can be made to scale with N instead of N^2. See chapter 6 for a detailed discussion on the implementation of these efficient algorithms.

3.4.2 Computer time required

Even with these approximations, performing MD on systems of tens of thousands of atoms can be very time consuming. Standard MD packages like LAMMPS [22] and NAMD [23] are programmed to run in parallel on a cluster of many computers. To give a concrete set of numbers, the typical polymer simulation run in this thesis is split across 64 CPU cores and runs for 24 hours. Of course, with GPU computing, the same simulation can be performed on a desktop computer in the same amount of time, but that is a topic for chapter 6.

3.5 Application to polymers

3.5.1 Modeling polymers using MD

To model the phases of polymer systems efficiently in MD, the concept of treating each atom individually must be forgotten. A single *short* polymer might contain thousands of atoms. At most, given the computer resources available, just a few polymers could be simulated together like this. Periodic boundary conditions will not help here as the length scale of interest, the repeat length of the ordered phase of polymers, is going to be much larger than a box with just a few polymers in it.

However, polymers have the fractal scaling behavior previously discussed in subsection 2.1.6. So in a model where each bead in the simulation represents a number of monomers, say 10, the results at long length scales will be the same (scaled, of course) as those where every monomer is treated individually. This technique is called coarse-graining. When modeling polymers of

finite length, the method can only be stretched so far, of course. A entire polymer cannot be represented by a single bead and retain the same properties. The limit to which a polymer can be coarse-grained is determined by the *Kuhn length* [1].

Further details on the choice of pair and bond potentials to implement this coarse-grained polymer model are explained in section 4.3 and to some extent in chapters 5 and 7.

3.5.2 Modus operandi

Any MD simulation must be started given some initial condition. When studying the self-assembled phases of polymer systems, the choice of the initial conditions is simple. A random number generator is used to place polymers in the simulation box. The parameters of the simulation are then set and a number of integration steps run (usually anywhere from 20–50 million). In this manner, the polymers are given no preferential initial state, leaving only the physics of the self-assembly process to drive the final phase to form.

Saving the state of the system at every single time step during the simulation is not feasible. A single simulation run could easily consume terabytes of hard disk space. Instead, snapshots are typically saved at regular intervals of 100,000 time steps or more. This is usually sufficient to determine the phase of the system and to provide some idea of the dynamics. When a more detailed accounting of the dynamics over short time scales is needed, particular simulations are rerun or continued with snapshots saved more often.

Thermodynamic state variables like temperature and the internal energy are also calculated and saved every so often. While these usually do not enter into the analysis of the polymer simulations presented in this thesis, they serve as good cross checks that the simulation ran correctly and without any strange behavior.

3.5.3 Equilibration

An understanding of equilibration in the context of MD is required in this thesis. After all, in the study of phase diagrams of polymer systems it is the equilibrium state that is the desired result of any simulation run. The principal of thermal equilibrium in statistical

mechanics states that any system, when left alone for a *long enough* time will reach a state of equilibrium. It is the *long enough* part that can become tricky to deal with in molecular dynamics.

So given a particular simulation, the question "has been it run long enough to reach equilibrium?" must be asked. In short, it is impossible to answer that question with a proof. One can, however, look at a number of indicators. When dealing with phases of polymer systems, the simplest way to check for equilibration is to play a movie of the entire saved simulation trajectory. If the simulation reaches a point where the resulting phase is stable and unchanging over a long time, then this is a good indicator that thermal equilibrium has been reached. Quantifiable parameters related to the structure can also be calculated and plotted versus time where, again, a stable region at the end indicates equilibrium has been reached (see chapters 4 and 5 for uses of such indicators).

However, there is the possibility that the dynamics of the system are so slow, especially for systems at a low kinetic temperature, that it becomes *kinetically arrested*. When this is the case, examining the trajectory or any of the parameters of the structure will show a stable region because the dynamics of the system have stopped, not because it has reached a final thermal equilibrium. These cases can often be spotted by looking at particle diffusion rates.

Finally, the possibility also exists that an apparent stable state is reached that is not kinetically arrested but is still not the true thermal equilibrium. Metastable states like these can be compared to supposed more stable states quantitatively by performing free energy calculations. But such calculations are tricky to perform and best not done unless something of particular importance will be learned. More easily, the presence of metastable states can be identified by performing many simulations starting from different initial configurations. At least then an argument can be made that the state which shows up more often is *likely* to be the true stable equilibrium. As a particular example, chapter 5 deals with a very challenging system that requires both a very careful choice of temperature to avoid kinetically arrested states and the right periodic box size to attain true equilibrium.

3.6 Units

A system of units has so far deliberately not been mentioned. After all, numbers stored and calculated in a computer are dimensionless, so the formulation of MD must not depend on the system of units. In fact, especially in the field of coarse-grained polymer simulations, it is most convenient to work in a system of reduced units defined by the pair potential. Two fundamental units come directly from the Lennard-Jones potential. First, the value of the potential is in units of energy and defined by the value of ε. In the computer ε is *effectively* set to one, but the results of a simulation are valid for any system of units simply by assigning a different value to ε and scaling the results. Similarly, the fundamental unit of length is defined by the pair potential as σ. And finally, when calculating the acceleration from a derivative of the energy (force), the mass of the particle m makes the third fundamental unit. As with the energy, these two values are effectively set to 1 in the computer but can be assigned a value to scale the simulation into any real system of units.

Given the three fundamental units of energy (ε), length (σ) and mass (m) any other units can be derived. For example, the unit of time $\tau = \sqrt{m\sigma^2/\varepsilon}$. Being concrete for the moment, if SI units of Joules, meters and kilograms are used, then the unit of τ would be seconds. Temperature deserves special mention. Instead of trying to convert Boltzmann's constant k_B into the system of reduced units, it is factored entirely out of every equation involving temperature. The reduced temperature $T^* = k_B T/\varepsilon$ is used in its place, and as a ratio of energies it is dimensionless.

CHAPTER 4. COARSE-GRAINED SIMULATIONS OF GELS OF NON-IONIC MULTI-BLOCK COPOLYMERS WITH HYDROPHOBIC GROUPS

A paper published in *Macromolecules* [1] [24]

Joshua A. Anderson and Alex Travesset

4.1 Abstract

Solutions of multi-block non-ionic polymers with hydrophobic blocks in water exhibit crystalline and liquid-crystalline phases over a narrow temperature range. This strong temperature sensitivity, critical in the design of novel self-assembled materials, is the result of the drastic increase of hydrophobicity combined with the weakening of solvating interactions (hydrogen bonding or dipolar) as the temperature is raised. In this paper we separate thermal fluctuations into a 'kinetic' temperature and solvation effects and parametrize temperature variations with a single parameter α, where the solvent is modelled implicitly. We provide a microscopic interpretation for this parameter and molecular dynamics simulations are used to investigate the phases of short ABCBA pentablocks, where the A and C blocks are hydrophobic and the B blocks are hydrophilic but contain hydrophobic groups. At low temperatures and for increasing concentrations, the system undergoes a sol-gel transition. The gel is swollen and consists of highly interconnected spherical micelles with a finite lifetime. At higher temperatures, lamellar and perforated lamellar phases are found for increasing polymer concentrations, while for intermediate concentrations the system is found in a supercoiled gel. We find good agreement

of our results with modified and inverted Pluronic® systems, and discuss the relevance for other polymers including hydrophobic blocks like telechelic or peptide based polymers.

4.2 Introduction

The success of self-assembly as a technique to make new materials hinges on our ability to synthesize or recognize the appropriate components, ranging from simple molecules to complex aggregates, and to identify the environmental conditions that must be met for the components to self-assemble into a material exhibiting the specific physical, chemical and/or biological properties of interest [25, 26]. Polymers offer unique properties as components [27, 28]. Ionic polymers (polyelectrolytes) are soluble in water and play a critical role in many biological and industrial processes. Non-ionic polymers may also be water-soluble if they contain functional groups that form sufficiently strong hydrogen bonds with water molecules.

Polymers in solution interact with each other leading to spatial and/or temporal organizations such as crystals, liquid-crystals, thermoreversible gels, etc. Our current understanding of these organizations and their dependence on chemical composition, chain topology, solvent properties and other factors is still poor [29]. Multiblock copolymers offer an even wider range of phases and morphologies. Telechelic polymers are ABA triblock polymers where the B block is a long water soluble chain and A blocks are short, surfactant-sized hydrophobic chains. Telechelic polymers in solution form flower-like micelles, which are well understood theoretically [30–32], and are responsible for the relative simplicity of its rheological properties [33]. Pluronic® polymers are short (about 100 monomers or less) triblock ABA polymers where A is polyethylene oxide (PEO) and B is polypropylene oxide (PPO) (see [34] for a recent review). The phase diagram of Pluronic® polymers as a function of temperature, polymer concentration, chain length and relative fraction of PEO to PPO monomers is fascinatingly diverse, and encompasses all phases observed in diblock copolymer melts [33, 35] and many others, such as nematic and cubatic liquid crystalline phases [33].

Pluronic® and telechelic polymers are just two examples exhibiting the diversity of phases and morphologies of solutions of multiblock polymers that combine both hydrophilic and hy-

drophobic blocks. The synthesis of novel multiblock copolymers and surfactants that self-assemble into new phases and morphologies has become a very active field. Recent examples include toroidal supramolecular assemblies [36], Y-junctions micelles [37] and dendritic surfactants showing a number of phases that include an A15 lattice [38]. This A15 lattice is characteristic of a minimal area tiling of the three dimensional space [39], and it has recently been predicted [40] to be a stable phase in melts of branched copolymers. Another example of polymer synthesis is the attachment of 2-(diethylamino)ethyl methacrylate (DEAEM) blocks to the ends of a Pluronic® F127 polymer [8]. DEAEM blocks are hydrophobic at high pH (pH $\geq$ 8), but as the pH is lowered the DEAEM groups gain a proton, become charged and therefore hydrophilic [41]. The combined Pluronic®-DEAEM system (PLD) is both strongly temperature and pH sensitive, where the strong temperature sensitivity is inherited from the Pluronic® chain and the pH sensitivity results from the DEAEM end blocks. This rich sensitivity to external changes allows the investigation of self-assembled structures by precisely and independently tuning the relative strength of hydrogen-bonding, electrostatic and hydrophobic forces.

In this paper, we provide a general model to investigate non-ionic multi-block polymers in solution where one or several blocks contain hydrophobic groups, and present concrete results for the PLD system for which abundant experimental data exists [8–10]. Previous MD simulations of the PLD system have been performed with a model where the solvent is considered explicitly [42]. Such simulations are appropriate to explore the structure of single micelles, but are still computationally too costly for determining phase diagrams. Implicit solvent models enable simulations of large systems [43–45]. Although several effects may not be reliably accounted for, most notably hydrodynamics, implicit models are expected to provide an accurate description of static equilibrium properties.

4.3 Model

4.3.1 Temperature sensitivity of hydrophobic monomers

A common characteristic of soluble polymers containing hydrophobic blocks is their strong temperature sensitivity. In Pluronics®, for example, different phases are observed over the narrow temperature range between 10-60^{o}C, with more ordered phases found for *increasing* temperature [46].

The origin of this anomalous behavior results from the drastic increase in hydrophobicity combined with the weakening of hydrogen bonding interactions, which enhance solubility, as temperature is increased. The temperature dependence of the hydrophobic effect can be seen in the analysis of the Ostwald coefficient for methane [47]. As shown in Figure 4.1 an increase in temperature of 35^{o}C, in absolute temperatures $\frac{\Delta T}{T} \sim 10\%$, results in an increases of hydrophobicity by a factor of two. Longer alkanes follow similar trends [47], so we assume a similar hydrophobic effect for polymers containing hydrocarbons. In Pluronics® polymers, the oxygen in the PEO groups has a strong hydrogen bonding interaction with water molecules, which is enough to keep PEO soluble at all temperatures [29]. In PPO monomers, however, the additional methyl group (see Figure 4.2) is enough to drive them insoluble at around 10^o C.

The consequence of Figure 4.1 is that the quality of the solvent (water) for hydrocarbons is rapidly decreasing over a narrow temperature range. Coarse-grained models for polymers containing hydrocarbon groups where the solvent is modelled implicitly or with a simplified potential must consider potentials with a strong temperature sensitivity. Otherwise, solubility will inevitably increase with temperature.

The phase diagram of an aqueous solution of PEO as a function of temperature and concentration is complex and has been the subject of recent studies [29, 48]. In this paper, we consider short PEO blocks at temperatures within the range $10 - 80$C. In this region of the phase diagram PEO blocks are well described as a chain in a good solvent.

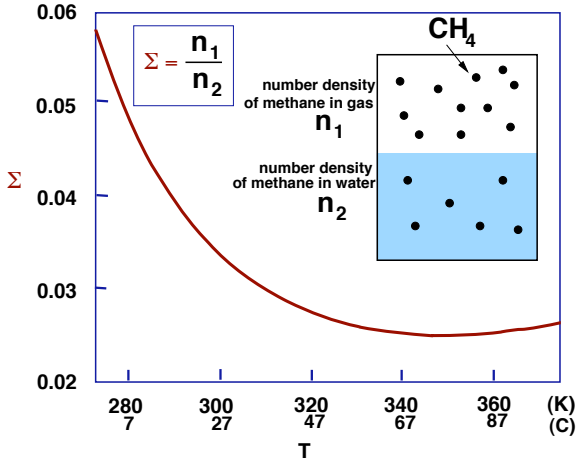

Figure 4.1 Ostwald coefficient Σ of methane as a function of temperature ([47] and references therein) showing the drastic change in solubility as a function of temperature.

4.3.2 Model and simulation details

The polymers we consider in this paper are pentablock copolymers with an ABCBA structure. As shown in Figure 4.2 each A block is composed of 6 beads and has two branches, the B blocks consist of 10 beads and the C blocks of 7 beads; each polymer contains a total of 39 beads. The choice of 10 Kuhn monomers for block B is consistent with a chain of 100 PEO units as it follows from experiment values [49]. The number of Kuhn monomers of the other blocks are then fixed by mimicking the composition of the real PLD system [8] as illustrated in Figure 4.2.

We describe the non-bonding interactions as effective Lennard-Jones 6-12 potentials. Following the discussion at the end of the previous subsection, the beads in both A and C blocks are hydrophobic[2] and interact with a potential

$$U_{AA,AC,CC}(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right]. \tag{4.1}$$

PEO monomers are soluble in water but the quality of the solvent is strongly temperature dependent. This property is modelled by a parameter α ($0 \leq \alpha \leq 1$) with the solvent becoming

[2]The temperature dependence for these blocks as described by Figure 4.1 is unnecessary as it is assumed that they are already dehydrated; that is, they avoid the solvent

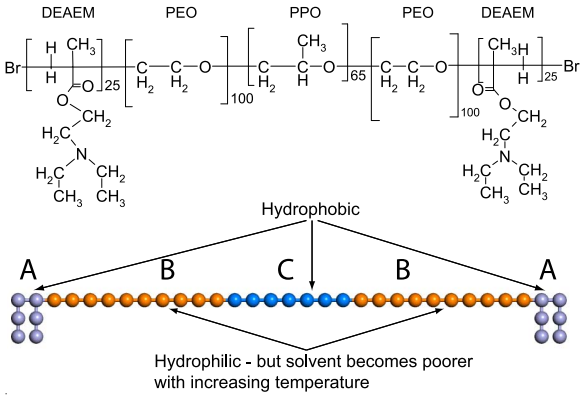

Figure 4.2 The PLD system consists of a F127 pluronic with DEAEM groups attached at both ends (adapted from [10]). Below we show the model representation used in the simulations. At high pH, DEAEM are hydrophobic.

of poorer quality with an increasing value of α. Thus the interaction of B with all beads is

$$U_{BA,BB,BC}(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \alpha \left(\frac{\sigma}{r}\right)^{6} \right] . \tag{4.2}$$

At $\alpha = 0$ this potential is purely repulsive

$$U(r) = 4\varepsilon \left(\frac{\sigma}{r}\right)^{12} , \tag{4.3}$$

so a given B bead will avoid all other non-bonded beads and prefer contact with the solvent. At $\alpha = 1$, the interactions of B are the same as with A and C and therefore hydrophobic. It should be noted that a similar model has been recently used to investigate the globular to coil transition [50]. All beads in our model have the same mass m and interaction range σ. It has recently been shown [51], that the solvent significantly modifies the packing of the polymer as compared with the same situation without solvent. This effect becomes particularly important for solvents bulkier than water, like benzene, but even in this case, the effect is short-ranged (less than 1 nm) so it is reasonable to expect that it can be incorporated into the parameters of the coarse-grained description.

The hydrophobic effect is the driving force responsible for the different structures observed, and in our model the hydrophobic interaction is parameterized by α (see Equation 4.2). Within

the temperature range between $10^o - 45^o$C, thermal fluctuations are around $\Delta T/T \sim 10\%$ while hydrophobic interactions change by a factor of around two, as discussed in the previous section. In our simulations, we will keep the *kinetic* temperature fixed and assume that temperature variations are fully accounted for by the parameter α. From previous considerations, the error involved in this approximation should be of the order of 10%, which is most likely less than the error resulting from coarse-graining and treating the solvent implicitly.

The nearest beads within the polymer chain are connected by a harmonic potential given by

$$U_b(r) = \frac{1}{2} k_b \left(r - r_0\right)^2 , \tag{4.4}$$

with $k_b = 330\varepsilon\sigma^{-2}$ and $r_0 = 0.84\sigma$ to avoid bond crossing. The polymers are considered to be completely flexible without any bending potential.

Simulations are carried out with the DL_POLY [52] simulations package. A number N_{poly} of polymers are considered in a finite cube of length L with periodic boundary conditions. The monomeric packing fraction ϕ_P is related to L and N_{poly} by

$$\phi_P = \frac{\pi N_{poly} N_{mon}}{6L^3} , \tag{4.5}$$

where the number $N_{mon} = 39$ is the number of beads per polymer. All potentials are cut-off at a value $r_c = 3\sigma$. Simulations are performed in the canonical NVT ensemble using the Nose-Hoover thermostat [15] for temperature control. The equations of motion are integrated using the Verlet leapfrog algorithm with a time step $\Delta t = 0.00203\tau$, where $\tau = \sqrt{m\sigma^2/\varepsilon}$ is the unit of dimensionless time.

The initial configuration is generated using the GenPol package [53]. GenPol uses a combination of Monte Carlo and Molecular Dynamics algorithms to create relaxed homogeneous polymer chains.

In the simulation runs of the swollen gel phase, the system is equilibrated for one million time steps. After the equilibration period, the configuration of the system is recorded every $20,000$ time steps as it runs for an additional four million. Figure 4.3 shows the typical behavior of the potential energy and the number of micelles during these runs. The number

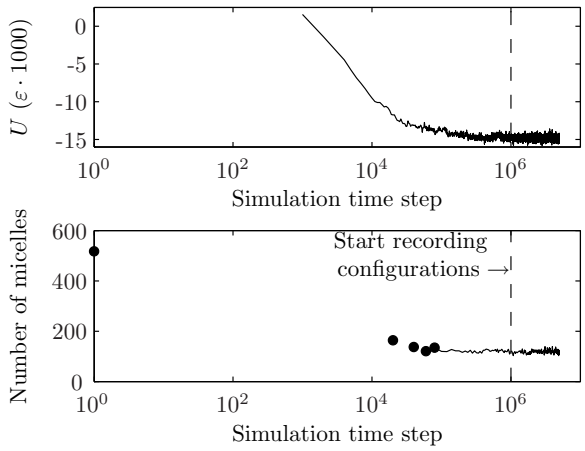

Figure 4.3 Typical time evolution of the configuration energy and number of micelles during the simulations. Configuration data needed to calculate the number of micelles is only sampled every $20,000$ time steps, so markers are drawn for the first few data points.

of micelles is given as a quantity more sensitive to the structure than the potential energy from which equilibration at one million time steps is clearly demonstrated. Judging from the micelle lifetimes, (see Figure 4.11 and containing section) a very conservative lower bound of the number of independent configurations for a single simulation run is about three. Considering that each configuration contains ≈ 200 micelles, that already gives a total of ≈ 600 independent micelles from which to generate statistics. We recall that in the micelle lifetime movie (see the supporting information), it is clearly seen that micelles change drastically in shape and size in less than half their lifetime, so the actual number of independent configurations is expected to be significantly larger than three.

The concentrated regime equilibrates *much* more slowly, and these simulation runs are usually equilibrated after 12-15 million timesteps. Overall, the total CPU time employed to run the simulations reported in this paper is of around 2.2 years on a single 3.2GHz Intel Xeon processor.

4.3.3 Observable quantities

The system investigated here generally forms micelles whose structure and superstructure we characterize with various observables. The first step is to identify individual micelles, and we proceed with a similar criteria used previously by other authors [45, 54]: Any A or C bead that lies within a distance r_{cut} of any A or C bead belonging to a micelle is included in that micelle. In order to identify micelles for a fairly large numbers of polymers (N_{poly} of the order 1000) an efficient algorithm is required. Consider the set of A and C monomers to be a fully connected graph. Each bead is a node in the graph and each edge has a weight equal to the distance between the beads. Then edges longer than r_{cut} are removed. Now, finding the beads belonging to each micelle is equivalent to finding the largest set of spanning trees that cover a partitioning of the graph. This can be done using a bottom-up algorithm with running time $\in \Theta(n^2)$ [55] where n is the number of A and C beads. The identified micelles are insensitive to r_{cut} from 1.22σ to 2.00σ. The value $r_{cut} = 1.44\sigma$ is used in all results presented in this paper.

Let N_{agg} be the aggregation number of a given micelle. With $\approx$ 200 micelles forming per configuration, the distribution of N_{agg} is of interest. Let $W(N_{agg})$ be the number of micelles with aggregation number N_{agg} in a given configuration, then the aggregation number distribution is found from

$$w(N_{agg}) = \frac{W(N_{agg})}{\sum_{N_{agg}} W(N_{agg})}. \tag{4.6}$$

The shapes of the micelles are characterized from the gyration tensor [54]

$$G_{\alpha\beta} = \frac{1}{N}\sum_{i=1}^{N}\langle(r_{i,\alpha} - R_{cm,\alpha})(r_{i,\beta} - R_{cm,\beta})\rangle \tag{4.7}$$

where the index i runs over all *core* hydrophobic beads within the micelle and $R_{cm,\alpha}$ is the center of mass of the micelle. The three eigenvalues of the matrix $\mathbf{G}$ (ordered as $g_1 > g_2 > g_3$) characterize the radii of a given micelle.

Several observables are used to characterize the superstructures of micelles. We define a *bridging polymer* as a polymer that has hydrophobic blocks in at least two micelles. The bridging fraction f_B is defined as the number of bridging polymers relative to the total number

of polymers. Percolation is also of interest, and we define the gel fraction f_{gel} as the fraction of polymers belonging to the largest group of micelles interconnected by bridging polymers.

We also investigate the statistics of the coordination number N_C of a micelle, defined as the total number of different micelles that a given micelle is connected to via bridging polymers. The distribution of coordination numbers $w_C(N_C)$ is defined in the same way as $w(N_{agg})$.

Ordered structures are analyzed by computing the static structure factor,

$$S(\vec{q}) = \frac{1}{N_{poly}N_{mon}} \sum_{i=1}^{N} \sum_{j=1}^{N} \langle e^{i\vec{q}(\vec{r_i}\cdot\vec{r_j})} \rangle, \tag{4.8}$$

and by visual inspection of the configurations.

Aggregation numbers are variable in time as micelles gain or lose polymers. We thus define the micelle lifetime τ_L as the time taken for a given micelle to lose half of its initial polymers. This is calculated as follows: Let $\mathbf{M}_j^i$ be the set of *core* hydrophobic monomers of micelle j at time step i. For each recorded time step k the set intersection $\mathbf{I}(j,k,m) = \mathbf{M}_j^0 \cap \mathbf{M}_m^k$ is calculated for all micelles m in time step k (time step 0 refers to the first timestep after equilibration). The micelle $m = m_{max}$ that gives the largest number of elements in $\mathbf{I}(j,k,m)$ is identified as the "same" micelle as j. The lifetime τ_L is then defined as the first timestep where the number of elements in $\mathbf{I}(j,\tau_L,m_{max})$ is less than half that in $\mathbf{M}_j^0$.

Solvent distribution is also of interest. In our model, solvent is implicit, so we compute the solvent distribution from the following; Spheres of radius 3σ are placed on a grid in the simulation box, and local distributions $\phi_{P,X}$ for $X = A, B, C$ beads are calculated inside each sphere. The solvent distribution $\phi_{P,S}$ is then obtained from $\phi_{P,S} + (\phi_{P,A} + \phi_{P,B} + \phi_{P,C}) = c$. The constant c defines the packing fraction and is chosen by assuming that "solvent" beads are absent in regions of high hydrophobicity, where only A and C beads are present. The value $c \approx 0.57$, which corresponds to random loose packing, was consistently used.

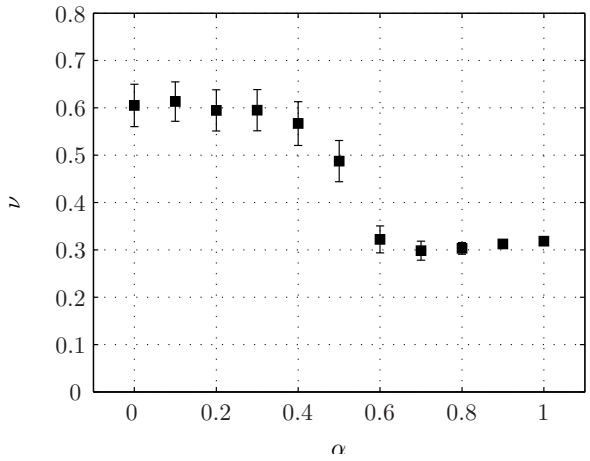

Figure 4.4 Flory exponent ν as a function of the parameter α for long PEO chains (composed of B beads only). Error bars are displayed at $\pm$ one standard deviation.

4.4 Results

4.4.1 The α parameter and the assumption of constant kinetic temperature

It is assumed that A and C blocks are hydrophobic and B blocks hydrophilic, but with the solvent becoming of poorer quality for increasing temperature. We provide a more quantitative mapping of the parameter α to the quality of the solvent by simulating a single polymer chain consisting only of B beads and computing the Flory exponent ν (by extrapolating the gyration radius as a function of increasing number of Kuhn lengths) as a function of the parameter α. The results in Figure 4.4 show a Flory exponent corresponding to a good solvent $\nu \approx 3/5$ in the region with α from 0–0.4, a theta point $\nu \approx 1/2$ around $\alpha \approx 0.45$ and poor solvent $\nu \approx 1/3$ in the region $\alpha \geq 0.45$. In simulations of PLD systems, where the B blocks are PEO and hydrophilic at all temperatures, the values of α are restricted to $\alpha < 0.4$.

We further assume that the dominant effect of the temperature is parameterized by α while the kinetic temperature is kept fixed. We briefly examine this assumption with the coarse-grained PLD model, shown in Figure 4.2; $w(N_{agg})$ is calculated at fixed α and ϕ_P and varying values of the kinetic temperature. Generally, large aggregation numbers are entropically unfa-

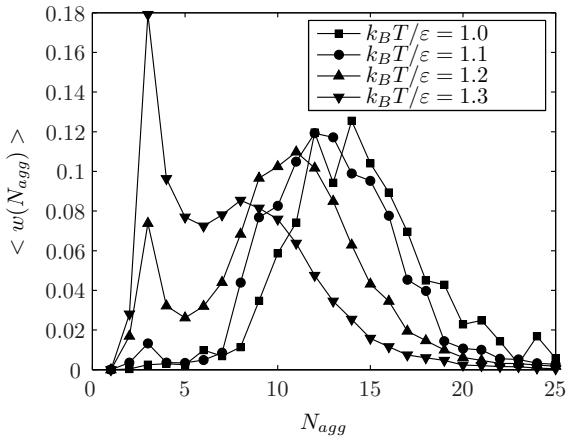

Figure 4.5 Average micelle aggregation number distribution at $\alpha = 0.0$, $\phi_P = 0.08$ and at various kinetic temperatures.

vorable, and smaller aggregation numbers should be expected for increasing temperature. The results in Figure 4.5 confirm this tendency and show that thermal fluctuations are not significant as k_BT/ε ranges from 1 to 1.2, but they become quite significant at $k_BT/\varepsilon = 1.3$, shifting to considerably lower aggregation numbers. We therefore interpret $k_BT/\varepsilon = 1.3$ as a critical temperature where thermal effects are dominant over the potential energy and will restrict the temperature within the domain $k_BT/\varepsilon < 1.3$. We recall that if we associate $k_BT/\varepsilon = 1$ to a temperature around 10^oC, $k_BT/\varepsilon = 1.3$ is a temperature of around 90^oC, so the approximation of constant "kinetic" temperature allows to explore the relevant temperature range.

4.4.2 The swollen gel

If B blocks are strongly hydrophilic, $\alpha = 0$, micelles consisting of cores of A and C beads and a corona of B blocks are formed. At low polymer concentration ϕ_P, the micelles are spaced far apart and the system consists of almost independent micelles. As ϕ_P is increased, micelles move closer together and become physically connected by bridging polymers. At a critical concentration, all micelles are connected and the system is a gel.

The gelation order parameter is $f_{gel} = 1.0$, indicating that polymers are connected (percolate) through the entire solution [1]. The critical gelation concentration ϕ_P^c (the lowest polymer

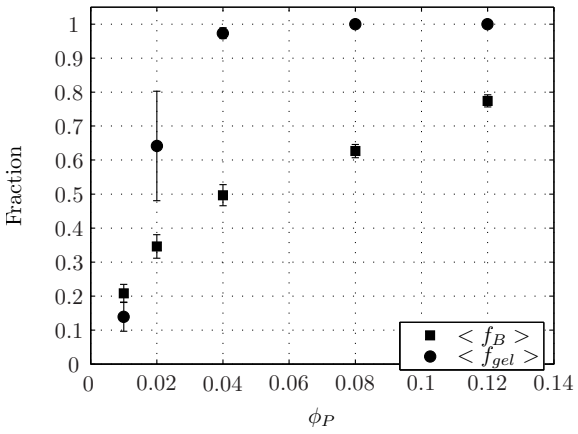

Figure 4.6 $< f_B >$ and $< f_{gel} >$ as a function of concentration at $\alpha = 0.0$. Error bars are displayed at $\pm$ one standard deviation. A sharp transition in both $< f_{gel} >$ and $< f_B >$ is observed from $\phi_P = 0.02$ to $\phi_P = 0.04$.

concentration where $f_{gel} \approx 1.0$) is obtained by inspection of Figure 4.6 and is $\phi_P^c \approx 0.04$. Examination of the snapshot in Figure 4.7 corroborates this; the gel is swollen and is composed of roughly spherical micelles. The sol-gel transition is very sharp, as at $\phi_P = 0.02$ the system is deep in the sol phase, while $\phi_P = 0.04$ is already in the gel phase. Also plotted in Figure 4.6 is f_B, the fraction of polymers bridging different micelles. This number is remarkably high, and for example, at the critical gelation concentration $\phi_P^c \approx 0.4$ only half of the polymers are fully contained within a single micelle. For higher concentrations, polymers fully contained in a single micelle become increasingly rare. The results in Figure 4.6 and Figure 4.7 are indicative of a swollen gel consisting of highly interconnected micelles. These results are qualitative similar to previous studies in triblock colpolymers [57].

The average micelle aggregation number distribution is shown in Figure 4.8 for various concentrations ϕ_P within the range 0.04–0.12. As the concentration is increased, the distribution is shifted slightly to larger aggregation numbers, yet the number density of micelles increases, as shown in Figure 4.9. Actual micelle number densities are only slightly less than what would be expected if N_{agg} remained constant. This shows the relatively minor effect that concentration has on micelle sizes.

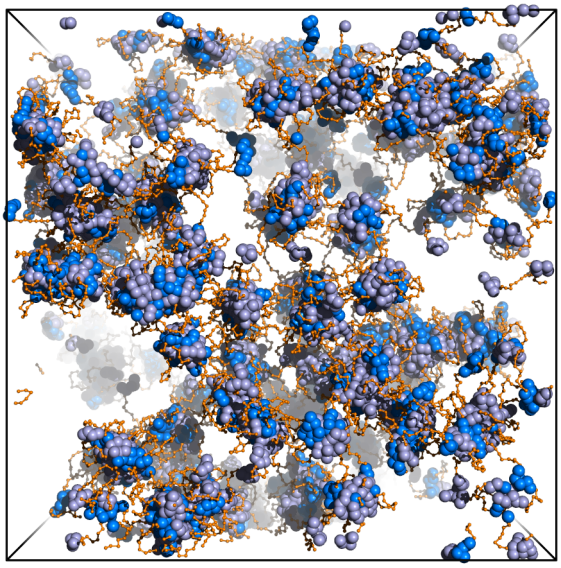

Figure 4.7 Snapshot of the simulation in the swollen gel phase at $\phi_P = 0.04$. A beads are colored light blue, B orange, and C dark blue. The A and C beads are drawn with a radius of 0.8σ. B beads are shown smaller so that the view of the micelle core structure is not obscured. Lines also connect the B monomers showing the bonds between beads in the polymer. Lastly, the micelles in the background fade out rapidly so that those in the foreground may be seen more clearly. All snapshots are created with PyMOL [56]

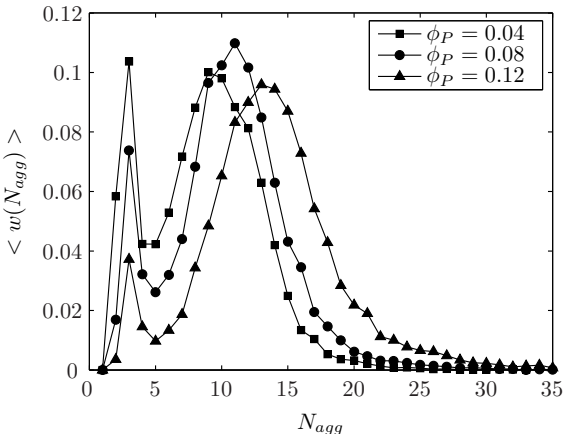

Figure 4.8 Average micelle aggregation number distribution at $\alpha = 0.0$ for various concentrations.

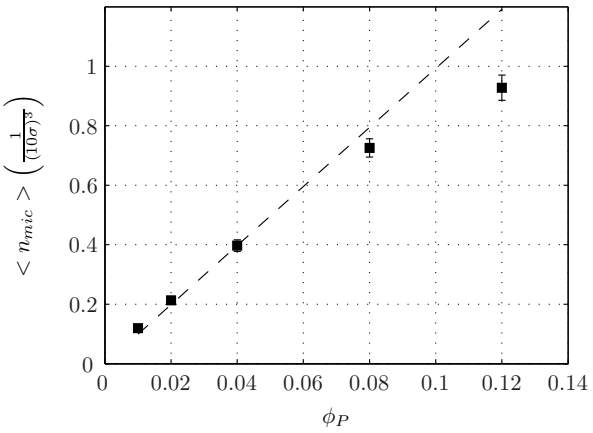

Figure 4.9 Number density of micelles as a function of concentration at $\alpha = 0$. The dotted line shows what is expected if $\langle N_{agg} \rangle$ at $\phi_P = 0.04$ were to remain constant for all ϕ_P.

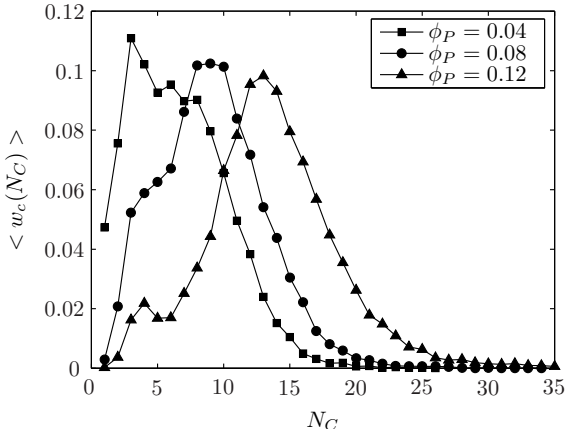

Figure 4.10 Average distribution of micelle coordination numbers at $\alpha = 0.0$.

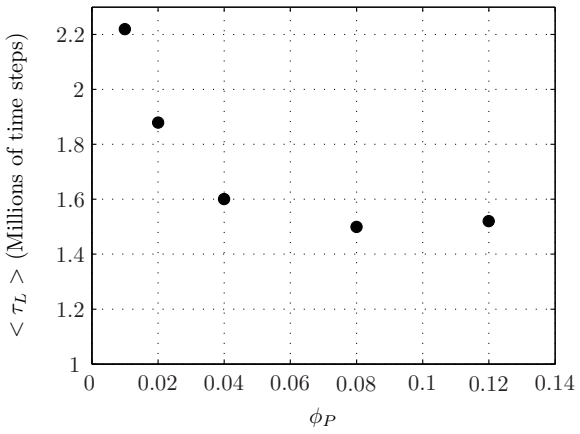

Figure 4.11 The average micelle lifetime as a function of concentration at $\alpha = 0$.

A rough idea of the superstructure of the micelles can be investigated from the average coordination number distribution $w_C(N_C)$, which is shown in Figure 4.10. N_C has a broad distribution that shifts toward higher values with higher concentration. The position of the peaks and even the shape of the distribution correspond closely with the aggregation number distribution in Figure 4.8. This is consistent with a given micelle coordinating with as many other micelles as it has polymers to do so. But $< f_B >$ from Figure 4.6 shows that *not all* of the polymers are bridging. This implies that there are a number of polymers bridging three micelles together with one hydrophobic block in each.

Thermal motion causes micelles to occasionally detach from the percolating gel, and consequently roughly four percent of the micelles at $\phi_P = 0.04$ actually have $N_C = 0$. This is in agreement with $\langle f_{gel} \rangle$ slightly less than one in Figure 4.6.

Ratios of the gyration radii provide a quantitative look at the shape of the micelles. For a perfect sphere, all ratios would equal 1.0. As seen in the snapshot (Figure 4.7), the micelles that form in our system are roughly spherical at $\alpha = 0$. The ratio of the longest radius to the shortest $\langle g_1/g_3 \rangle = 2.2 \pm 0.75$ and the middle to the shortest $\langle g_2/g_3 \rangle = 1.4 \pm 0.23$. Given that this data is calculated from a conservative estimate of ≈ 600 independent micelles, this spread (one standard deviation) is *not* the result of small statistics; instead, there is a wide variation

in micelle shapes. However, all are roughly spherical, as long cylinders would be characterized by a larger $\langle g_1/g_3 \rangle$.

Micelle lifetimes are shown in Figure 4.11 in the range $\phi_P = 0.01$–0.12. The free energy cost of having hydrophobic units in contact with the solvent is high, and therefore it should be expected that isolated micelles are very stable. If other micelles are near, however, a chain can bridge between micelles and eventually transfer from its original micelle to its neighbor by thermal motion. All these considerations are in agreement with the results in Figure 4.11, where the micelle lifetime is largest at low concentrations $\phi_P = 0.01$ and decreases with concentration in parallel to the increase in number density of micelles (see Figure 4.9). Rather interestingly, micelle lifetime shows a plateau at $\phi_P \geq 0.08$. This implies that there is some competing factor decreasing the rate of polymer transfer. Quantitative comparisons of micelle lifetimes and polymer transfer are strongly dependent on the nature of the solvent, so will not be considered in this paper.

The dynamical nature of the swollen gel is clearly visualized in the movie provided in the supporting information. From start to finish, this animation covers one million time steps of simulation time with frames spaced $5,000$ time steps apart. Beads are depicted in the same manner as in Figure 4.7 except that the polymers belonging to one micelle in the first frame are colored red. Some other polymers are colored green. The evolution of red polymers are observed as a function of time, and the dynamics of the micelle lifetime including polymer transfer processes are readily visualized. Creation of micelles are also visualized; The green polymers are initially well separated in space, but as a function of time they come together and form a new micelle.

The structure factor of the gel, which is provided in the supporting material, indicates a liquid-like structure with no evidence for any additional order, either crystalline or liquid crystalline.

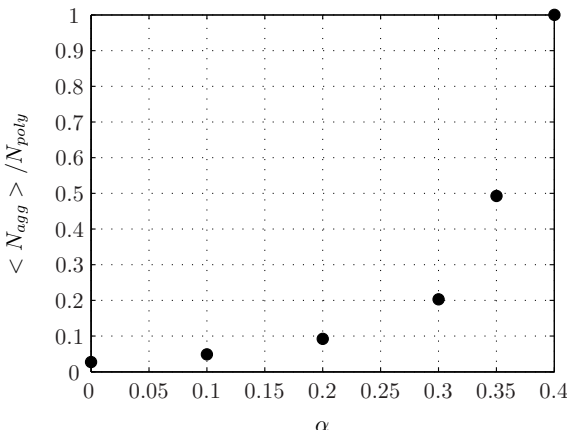

Figure 4.12 Average aggregation number of micelles for $\phi_P = 0.04$. In these runs, $N_{poly} = 600$.

4.4.3 The swollen to dry transition

We now investigate the properties of the gel as the B blocks become less hydrophilic by increasing α from 0.0–0.40, following the discussion from Figure 4.4. As α is increased, micelles are brought closer together as the B beads in the micelle coronas effectively attract one another, resulting in overall larger aggregation numbers and fewer micelles. As shown in Figure 4.12, the aggregation numbers steadily increases with α. The effect is small at first, but around $\alpha \sim 0.2$ aggregation numbers grow quite rapidly eventually collapsing all polymers in the simulation box into one giant micelle as α nears 0.40. A plot of micelle number density, shown in the supporting information, confirms the decrease of the number of micelle for increasing values of the α parameter with an almost linear dependence.

The characteristic size of the micelles is very sensitive to α. As α is increased, micelles evolve from a roughly spherical shape to a more cylinder-like one. The ratios g_i/g_j are used to quantify this transition; Figure 4.13 shows the results. The large variation in shapes seen at $\alpha = 0$ remains for higher α but the trend for $< g_1/g_3 >$ increases dramatically. But even at $\alpha = 0.35$, $< g_2/g_3 >$, the ratio of the middle radius to the shortest, remains small implying a roughly circular cross section. These two taken together show relatively long micelles with roughly circular cross sections suggestive of cylindrical micelles.

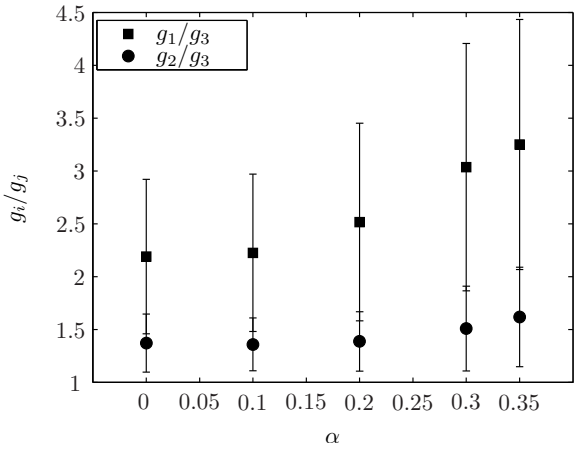

Figure 4.13 Gyration radii ratios as a function of alpha. Results for $\phi_P = 0.04$ shown, but there is little difference for ϕ_P from 0.04–0.12. Error bars are shown at one standard deviation.

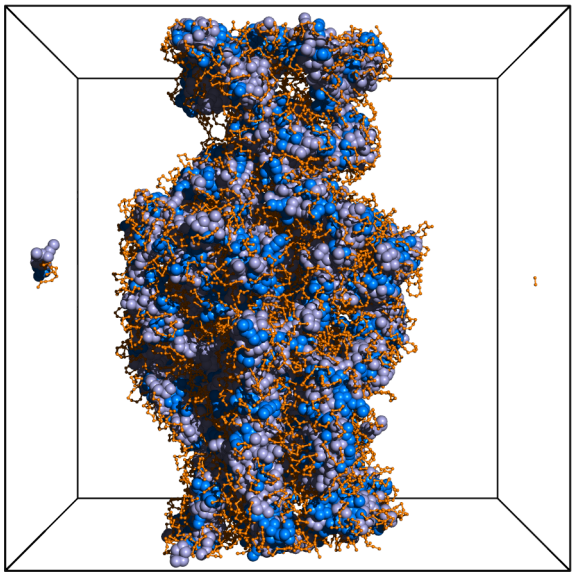

Figure 4.14 Snapshot of the simulation at $\phi_P = 0.04$ and $\alpha = 0.4$ showing a cylindrical micelle.

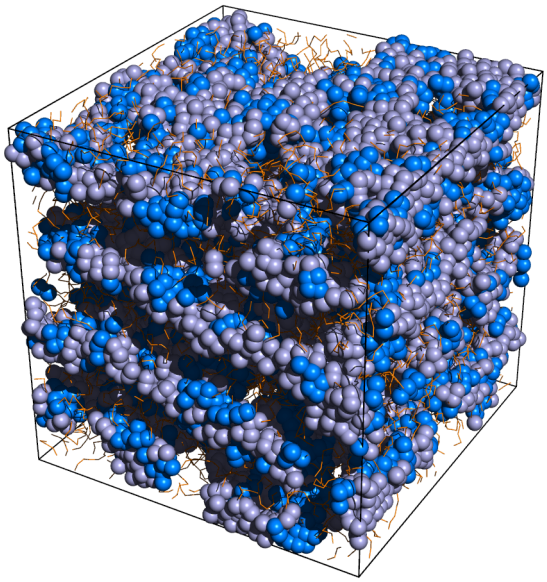

Figure 4.15 Snapshot of the system at $\phi_P = 0.32$ and $\alpha = 0.35$. Visible perforations are found across the lamellar planes.

A snapshot of the system at $\alpha = 0.4$ is shown in Figure 4.14. Visual inspection indeed shows cylindrical worm-like micelles. Their cores consist of A and C beads with B beads in the corona. These worm-like micelles bunch together so that many B beads are not even in contact with the solvent. Figure 4.14 also shows the presence of giant micelles at $\alpha = 0.4$, that is, micelles that wrap around the simulation box. We will further discuss the significance of these micelles in the next section.

Results at slightly higher concentrations $\phi_P = 0.08$–0.25 show similar trends, with a swollen gel phase of roughly spherical micelles for $\alpha \approx 0$–0.25 and giant cylindrical micelles for $\alpha \approx 0.3$–0.4. The analysis of the structure factor $S(\vec{q})$ shows a liquid-like structure for the swollen gel and no particular order within the giant cylindrical micelles.

4.4.4 The concentrated regime

The concentrated regime reveals a remarkable degree of order. At polymer concentrations $\phi_P = 0.30$ and $\alpha \sim 0.35$, a perforated lamellar structure is observed in visual inspections of the configurations as shown in Figure 4.15. The structure factor $S(\vec{q})$ (shown in the supporting

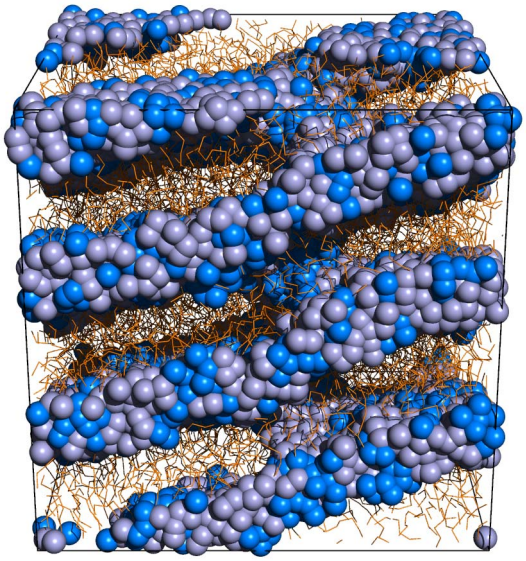

Figure 4.16 Snapshot of the system for $\phi_P = 0.50$ and $\alpha = 0.35$, forming a lamellar phase. A screw dislocation is clearly visible.

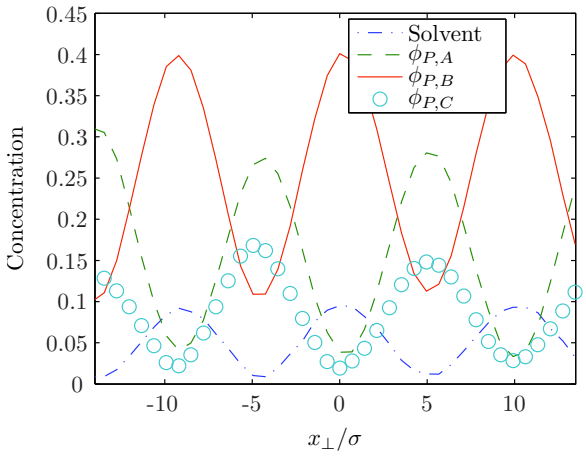

Figure 4.17 Concentrations of the solvent and A,B and C beads along the direction perpendicular to the lamellar planes.

material), confirms the lamellar structure. As the concentration is increased, the perforations eventually disappear leading to a regular lamellar structure. Both regular and perforated lamellar phases are stable against variations of the kinetic temperature $k_BT/\varepsilon < 1.3$. If the quality of the solvent is increased, the perforated lamellar phase gradually disappears. In good solvent ($\alpha = 0$), the perforated lamellar is absent and only the lamellar phase is found.

The structure of the lamellar phase consists of planes of A and C beads segregated from the B beads and solvent which can be seen from the snapshots in Figure 4.16. A quantitative analysis is shown in Figure 4.17, where the local distributions $\phi_{P,X}$ of the different beads and solvent are shown. The maximum of the solvent concentration coincides with the maximum of the hydrophilic B-blocks and the minimum of the hydrophobic A and C blocks. This lamellar structure is also corroborated from the analysis of the structure factor shown in the supporting material. From the peaks one extracts a lattice spacing of $\approx 10\sigma$, which is in perfect agreement with the thickness of each of the two planes obtained by inspection of the snapshots and from the analysis of the local distributions in Figure 4.17.

In almost all runs, the regular lamellar phase shows a screw dislocation, which is clearly visible in Figure 4.16. Only in one run corresponding to $\alpha = 0$ the lamellar phase did not show any topological defects. This may suggest that the elastic constants of the lamellar phase increase with the quality of the solvent.

4.5 Discussion

4.5.1 Phase diagram

The results in the previous section are summarized in Figure 4.18 and show a sol phase at low concentrations and good solvent for B blocks, followed by a swollen gel and a lamellar phase for increasing concentration. As the solvent for B-blocks becomes poorer, a giant micelle that wraps around the simulation box is observed, which is followed by a perforated lamellar and a lamellar phase as the concentration is increased. The points in Figure 4.18 represent the different simulations presented in parameter space. The perforated lamellar is not found for good solvent $\alpha \approx 0$, as it follows from the results at concentration $\phi_p = 0.3$ and other simulations,

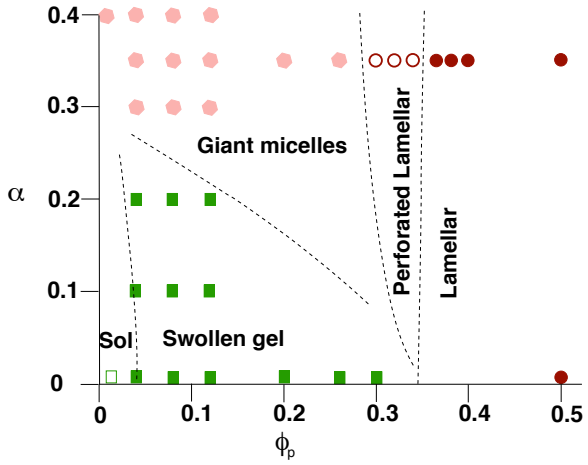

Figure 4.18 Summary of the results obtained in the previous section. The symbols are points of the phase diagram where actual simulations were run. The boundaries are drawn for a better visualization according to the discussion in the text and are not rigorous.

which although not fully thermalized (and therefore not shown in the graph) clearly indicated that the system remained in a swollen gel. We also expect that the critical concentration between the sol-gel transition should decrease with temperature as higher aggregation numbers are more easily formed. For the same reason, we also expect the giant micelles to become possible at lower temperatures for increasing concentration. This discussion is summarized in Figure 4.18.

The sol, swollen gel, perforated lamellar and lamellar correspond to phases that should be observed in experiments (a point we discuss further below). The region described as giant micelles, however, corresponds to large micelles wrapping around the simulation box, and are therefore a result of the finite simulation box and the periodic boundary conditions used in our simulations. At low concentrations, we interpret the results as implying roughly long worm-like cylindrical micelles. As polymer concentration is increased or solvent quality decreased, we expect these micelles to aggregate forming increasingly larger and more complex structures. On further increase of either solvent quality or concentration, we expect these aggregates to segregate (precipitate), forming what is known as a supercoiled gel [14]. The supercoiled gel

consists of a high polymer concentration with a low water content. For this composition, Figure 4.18 predicts a lamellar phase. It should be noted, however, that other processes may take place. For example, the supercoiled gel may result from condensation of cylindrical micelles or from aggregation of more disordered structures, with subsequent reorganizations after they precipitate. That is to say, the structure of supercoiled gels is not only dependent on the *final ensemble* but also on the *preparative ensemble* [14].

The expected phase diagram relevant for experiments is shown in Figure 4.19. The sol phase is a micellar liquid with some micelles clustering together. This is followed by a swollen phase. Solvent quality is parameterized by α, which is a monotonic function of temperature. So for increasing temperature we expect the micelles to grow in size leading to a liquid of cylindrical worm-like micelles and a supercoiled gel, depending on concentration. The determination of the structure of the supercoiled gel is beyond the scope for this paper. Finally, the lamellar phases are found at larger concentrations. The sol and swollen phase show large density fluctuations that decrease with concentration. Density fluctuations are small for the other phases except possibly at phase boundaries, which lie beyond the scope of this paper.

4.5.2 Comparison with experiments

Simulations results can be directly compared to PLD systems at high pH (pH $>$ 8), where the DEAEM block (A beads) are hydrophobic. There is abundant experimental data [8–10], which we compare to the theoretical results in this section.

In the dilute limit (2% in weight), SANS experiments at high temperatures (70^{o}C) show cylindrical micelles [10], while for lower temperatures (20^{o}C) spherical micelles are suggested [8]. In more concentrated regimes (20% in weight) a transparent, yet rigid gel phase is observed up to low temperatures ($\sim$ 7^{o}C) using tube inversion and rheological measurements [9]. Further increasing of the temperature above 45^o C a physical hydrogel, which we identify with the supercoiled gel, is formed. The formation of the hydrogel is very slow at these temperatures and proceeds more quickly at a higher temperature (70^oC). The melt state has also been investigated. Structural studies with SANS report evidence for a lamellar phase [10], although

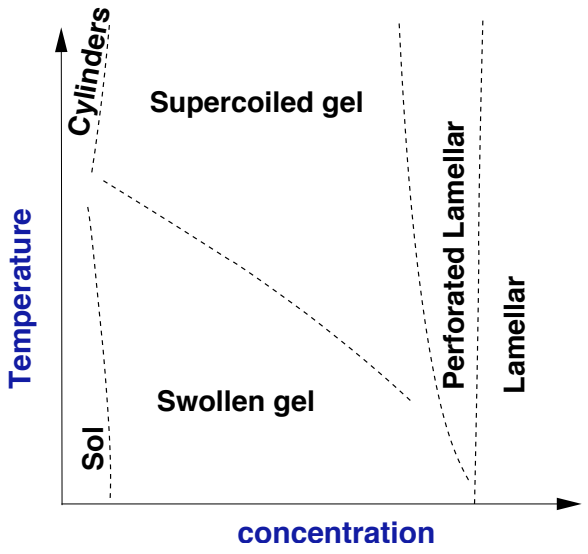

Figure 4.19 Phase diagram of the PLD system extrapolated from the results in Figure 4.18. We recall that the temperature and concentrations displayed in the plot are not a linear function of the temperatures and concentrations in the real system.

more experimental evidence will be needed. Equilibration times for both the melt state and the hydrogel are very long (of the order of 1 day). All our numerical simulations are in good agreement with these experimental results.

The structure of the hydrogel, which as stated, we associate with the supercoiled gel in our simulations, reveals a hexagonal lattice of close packed cylinders with a water content of around 30% in weight [10]. At this water content, our simulations would predict a lamellar phase, but this lamellar phase corresponds to a different preparative ensemble. We speculate that the hexagonal structure is the result of two steps, the formation of cylindrical micelles and the posterior self-assembly of these micelles into a hexagonal structure. Further work will be needed to clarify this point.

Inverted Pluronics® are triblock PPO-PEO-PPO polymers. On a qualitative level, they present some similarities with PLD systems, in that they contain hydrophobic end blocks in between overall hydrophilic blocks (PEO), and therefore show the same temperature sensitivity discussed for PLD systems. Phase diagrams for these systems have been reported and are

qualitative very similar to Figure 4.19. The observed phases are a random network (sol phase), a micellar network (swollen gel) and a uniform micellar phase, which we attribute to the supercoiled gel. At higher concentrations a lamellar phase [58] is also found. Furthermore, these phases follow the same sequence as a function of both concentration and temperature as the one in Figure 4.19. Many details regarding the boundaries and the critical concentrations separating these phases are quantitatively different, but we regard these results as providing additional evidence on the validity of our theoretical results.

4.6 Conclusions

In this paper we separated thermal effects into a "kinetic temperature" and solvation effects (hydrophobic, and to a lesser extent hydrogen bonding), which has allowed to parameterize the temperature by a single parameter (the α parameter) in an implicit solvent coarse-grained model. We determined the phase diagram of the PLD system both as a function of temperature and concentration and found a swollen and dry gel, as well as lamellar and perforated lamellar phases. The results are in good agreement with experimental results on PLD systems and on inverted Pluronic®. We provided a cross-check for the validity of our assumptions (Figure 4.5) and showed that the parameter α is a monotonic function of the temperature, which is well correlated with the Ostwald coefficient for alkanes Figure 4.1, which is a measure of the hydrophobic effect. We also provided a detailed characterization of the different phases involved. The model presented is general and is suitable to determine phase diagrams as a function of concentration and temperature for other systems such as Pluronic®, telechelic or peptide based and other non-ionic polymers that are soluble yet contain hydrophobic groups that decrease the solubility of the polymer for increasing temperature.

Explicit solvent models provide a realistic description of the dynamics and a more accurate description of the statics [42,59–61] but the CPU cost involved for simulating the large systems ($N_{poly} = 600$) investigated here would be very demanding. From the extensive experience gained in simpler models [1, 14, 62], we expect that the main conclusions of this paper will not be modified by the use of an explicit solvent. The system studied in this paper has some

connections with the HP copolymers studied recently by scaling methods [63]. We point out, however, that Pluronic® and related polymers are short (of the order of 10 Kuhn lengths or less) and scaling theories may not provide an accurate description for these systems.

There are many situations that we expect to discuss further in the future. The polymers investigated in this paper are slightly branched (the A-blocks). Recent results in melts show the existence of exotic phases for branched polymers [40]. It is our expectation that polymers with different branching degrees will include the phases found in melts but by tuning temperature, pH or concentration, additional phases will be found. The precise control and understanding of these phases provide exciting components for novel thermo-sensitive self-assembled materials. We hope that the simulations presented in this paper will stimulate further experimental work in this very exciting and relatively unexplored area.

Supporting Information Available In addition to figures and movies mentioned in the body of this paper, there are DIVX AVI clips of the snapshots. These rotate and/or rock the image to provide a better look at the structures. This material is available free of charge via the Internet at http://pubs.acs.org.

4.7 Acknowledgments

We acknowledge many discussions with S. Chushak, M. Determan, C. Lorenz, S. Mallapragada, T. Prozorov, W. Bu and D. Vaknin. We also thank C. Zaruba for technical assistance. This work has been supported by a DOE grant under contract W-7405-ENG-82 and partially supported by a NSF grant DMR-0426597.

4.8 Supporting Information

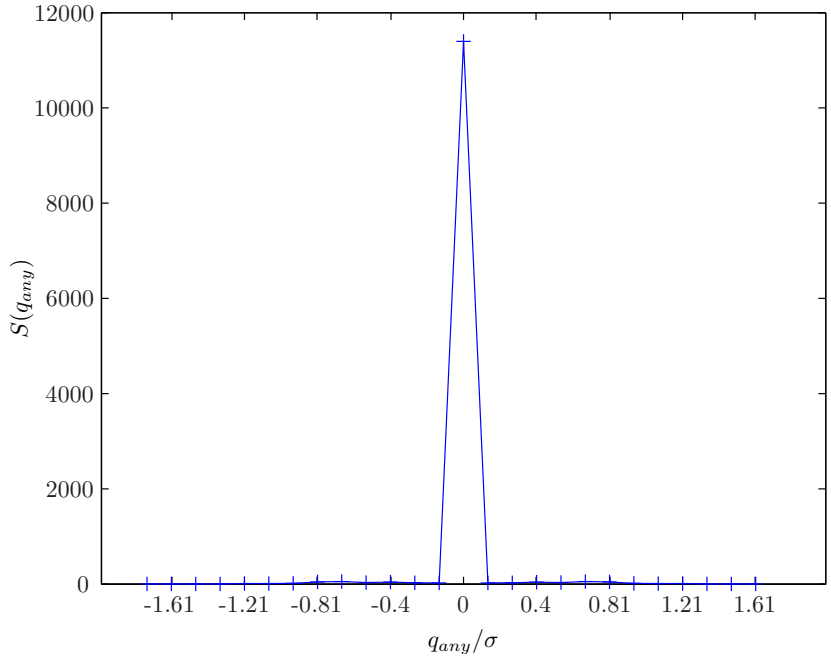

Figure 4.20 Structure factor calculated for the gel phase ($\alpha = 0.0$, $\phi_P = 0.04$); $\vec{q}$ is chosen along the z axis, but little difference in $S(\vec{q})$ is observed along any arbitrary direction. Notice the absence of any substantial peaks.

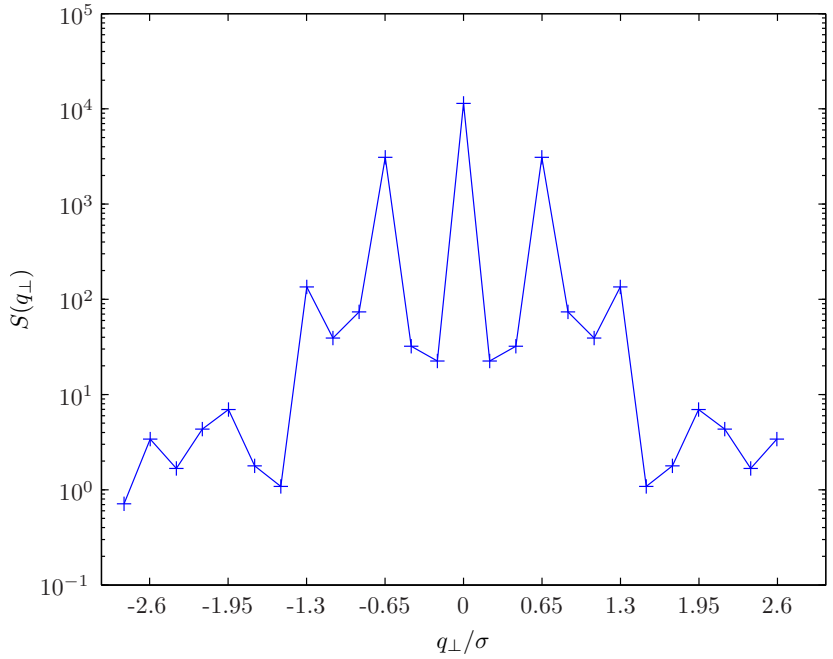

Figure 4.21 Structure factor calculated for the lamellar phase ($\alpha = 0.0$, $\phi_P = 0.50$); $q_\perp$ is along a line perpendicular to the lamellar planes.

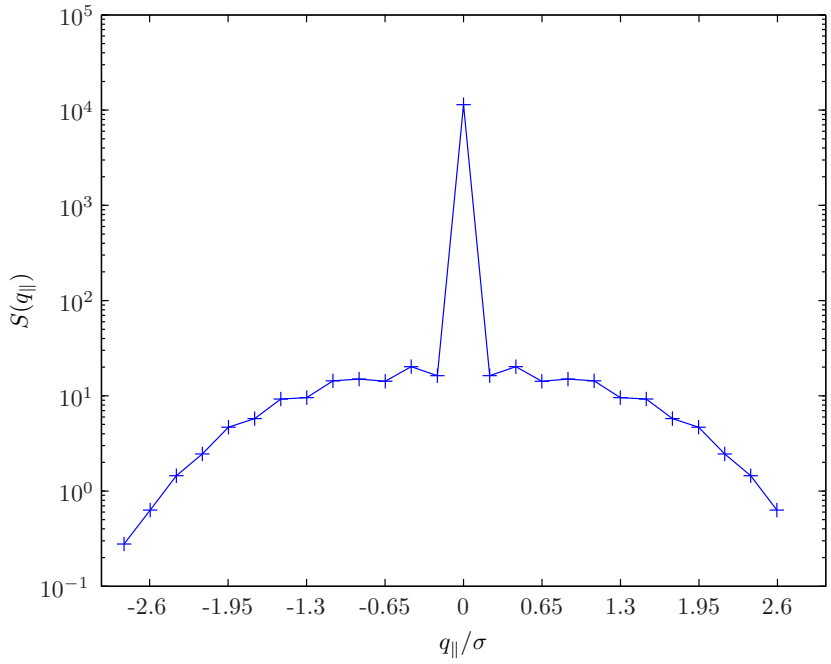

Figure 4.22 Structure factor calculated for the lamellar phase ($\alpha = 0.0$, $\phi_P = 0.50$); $q_\parallel$ is along a line parallel to the lamellar planes.

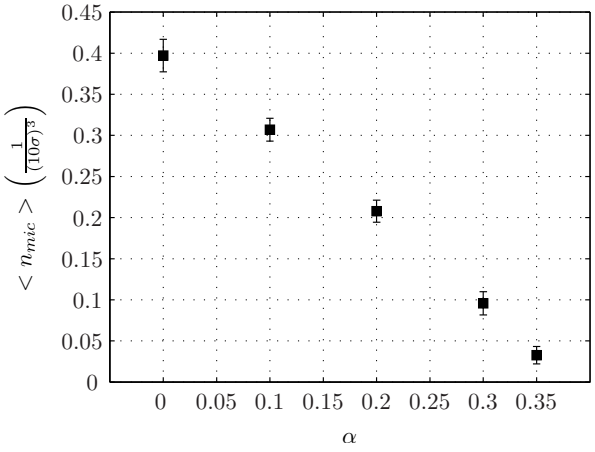

Figure 4.23 Number density of micelles for $\phi_P = 0.04$. The simulation runs include only 600 polymers showing that the total number of micelles decreases with α.

CHAPTER 5. MICELLAR CRYSTALS IN SOLUTION FROM MOLECULAR DYNAMICS SIMULATIONS

A paper published in the *Journal of Chemical Physics* [64]

Joshua A. Anderson, Chris D. Lorenz and Alex Travesset

5.1 Abstract

Polymers with both soluble and insoluble blocks typically self-assemble into micelles, aggregates of a finite number of polymers where the soluble blocks shield the insoluble ones from contact with the solvent. Upon increasing concentration, these micelles often form gels that exhibit crystalline order in many systems. In this paper, we present a study of both the dynamics and the equilibrium properties of micellar crystals of triblock polymers using molecular dynamics simulations. Our results show that equilibration of single micelle degrees of freedom and crystal formation occurs by polymer transfer between micelles, a process that is described by transition state theory. Near the disorder (or melting) transition, bcc lattices are favored for all triblocks studied. Lattices with fcc ordering are also found, but only at lower kinetic temperatures and for triblocks with short hydrophilic blocks. Our results lead to a number of theoretical considerations and suggest a range of implications to experimental systems with a particular emphasis on Pluronic polymers.

5.2 Introduction

Micelles of multi-block polymers are finite aggregates, typically around fifty polymers or less, where the insoluble blocks shield the soluble ones from contact with the surrounding solvent. Depending on control variables (temperature, polymer concentration, pH, etc..) micelles

may self-assemble into gels that exhibit long-range order such as bcc, fcc, hcp or other, more unusual, crystals. Micellar crystals exhibit a number of unique properties that have made them extremely attractive for fundamental studies as well as for applications [33, 65]. Extensively studied experimental systems include aqueous solutions of Pluronics (also known as Poloxamers), ABA triblocks where A is Polyethylene oxide (PEO) and B is Polypropylene oxide (PPO) [46, 66, 67] and inverted Pluronics, where the A blocks are PPO and the central block B is PEO [58, 67] as well as non-aqueous systems such as Polystyrene-Polyisoprene (PS-PI) diblocks in decane [68] and other solvents [69–72].

Micelles in solution are highly dynamical entities with polymers continually being absorbed and released through time. Therefore, a micellar crystal has a considerably intricate structure, where the long range order remains stable as the individual polymers are constantly hopping from one micelle to the next. Theoretical approaches such as density functional or mean field theory [39, 65, 73–77] directly study the ordered micelles and ignore the dynamical degrees of freedom of the polymers. Studies using molecular dynamics (MD) have the advantage of providing a reasonably realistic description of the dynamics, thus allowing the investigation of the role of single polymer degrees of freedom. In contrast with other approaches, MD also offers the important advantage that no assumptions need to be made about what is the possible thermodynamic state of the system.

The goal of this study is to predict phase diagrams of triblock polymers using MD simulations and to gain an understanding of the dynamics of micellar crystal formation. Because of our ongoing interest in Pluronic systems in aqueous solutions [24], we examine systems of $A_nB_mA_n$ triblocks, where the A beads are hydrophilic and B beads are hydrophobic. Although there have been a number of previous studies of multiblock copolymers in solution using MD, see Ref. [11] for a recent review, the prediction of crystalline structures presents substantial difficulties. Experimentally, it is well known that the approach to thermodynamic equilibrium is slow in these systems, with time scales of the order of minutes or hours. Therefore, even with suitably coarse-grained models, the long time scales involved provide a considerable challenge for MD simulation studies.

In this paper, we provide a detailed investigation of self assembled micellar crystals using MD. We aim to understand the mechanism by which they form, predict their range of stability and elucidate their static and dynamic properties. We also present an in-depth study on the challenges associated in reaching the thermodynamically stable state using MD and a successful strategy to overcome them.

5.3 Model and Simulation Details

5.3.1 Simulation Details

Systems of polymers are modeled by coarse-grained beads in an implicit solvent. Ref. [24] (chapter 4) provides a more detailed justification than the outline provided here. Individual polymers are $A_nB_mA_n$ symmetric triblocks, where A beads are hydrophilic and B beads are hydrophobic. All systems in this work are monodisperse with fixed values for n and m. Non-bonded pair potentials consist of a standard attractive Lennard-Jones potential for hydrophobic interactions

$$U_{\mathrm{BB}} = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] , \tag{5.1}$$

and a purely repulsive potential for hydrophilic interactions

$$U_{\mathrm{AA,AB}} = 4\varepsilon \left(\frac{\sigma}{r}\right)^{12} . \tag{5.2}$$

Pair potentials are cutoff to zero at $r = 3.0\sigma$. All beads have the same mass M. M, ε and σ are arbitrary and uniquely define the units of all numbers in the simulations. Neighboring atoms in the polymer chain are connected together with a simple harmonic potential. The packing fraction ϕ_P of the polymers is given by

$$\phi_P = \frac{\pi N_{\mathrm{poly}} N_{\mathrm{mon}}}{6(L/\sigma)^3} , \tag{5.3}$$

where N_{poly} is the number of polymers in a box of linear dimensions L and $N_{\mathrm{mon}} = 2n + m$ is the number of beads in each polymer. Simulation boxes are cubic with periodic boundary conditions. For this work, we use a time step of $\Delta t = 0.005\sqrt{m\sigma^2/\varepsilon}$. All simulations were

performed using the LAMMPS software package [22] in the NVT ensemble via Nosé-Hoover dynamics [17].

The temperature sensitivity of Pluronic systems is a reflection of the underlying strong temperature dependence of the hydrophobic effect. To model temperature dependent phases in such systems, the solvent quality for the A beads must change [24]. In this paper, we keep the solvent quality fixed and vary only the kinetic temperature when needed to equilibrate the systems properly.

Recently, a different implicit solvent model has been developed for describing Pluronic systems [78], where the pair potentials for the coarse-grained simulation are fitted to results obtained from all-atom MD and quantum chemistry simulations. Although the coarse-graining is different (each monomer A,B represents one PEO or PPO monomer) than in our model, the resulting potentials are quite similar to the ones in this work, Equations 5.1 and 5.2, with the only significant difference that the values of σ are different between the two A and B monomers, and U_{AA} shows a minor maximum.

5.3.2 Observables

A number of observable quantities are monitored for every recorded time step during a simulation run. The micelles themselves are identified by the same algorithm used previously in Ref. [24]. Any hydrophobic beads within a distance of $r_{\mathrm{cut}} = 1.2\sigma$ of one another are identified as belonging to the same micelle. Identified micelles containing less than 3 polymers are typically free polymers in the process of being transferred from one micelle to another and are removed from further consideration. Observables such as statistics of micelle aggregation number, gyration tensor, center of mass, and micelle lifetime are calculated and examined for every simulation performed in this work. Methods used to calculate these are described in Ref. [24].

The structure factor $S(\vec{q})$ is calculated over the center of mass coordinates $\vec{r}_i$ of all N_{mic} micelles in the system using the formula

$$S(\vec{q}) = C_0 \langle \left| \sum_{i=1}^{N_{\mathrm{mic}}} e^{i\vec{q}\cdot\vec{r}_i} \right|^2 \rangle \ , \tag{5.4}$$

with the components of $\vec{q}$ as multiples of $\frac{2\pi}{L}$ due to the use of periodic boundary conditions. The peaks in $S(\vec{q})$ are then used to reconstruct the full 3D real space lattice basis, if it exists. In this manner, $S(\vec{q})$ is not being used to simulate real scattering intensities as may be obtained by X-ray experiments, but as a mathematical order parameter to discriminate between the different ordered structures that may be present. For convenience, the normalization C_0 is chosen so that $S(\vec{q}=0)=1$. A more sophisticated treatment that allows continuum values of $\vec{q}$, suitable for quantitative comparisons with X-ray experiments, has been recently introduced [79], but we do not use it here.

Individual polymers are constantly hopping from one micelle to another. This is quantified over the entire simulation box as an overall rate of polymer transfer, r_{PT}, by examining contiguous simulation snapshots. Sets of indexed polymers belonging to each micelle are compared between the snapshots to find the number of polymers transferred. The rate r_{PT} is then expressed as a fraction of the number of polymers in the box transferred per one million time steps. A polymer that is transferred out and back to the same micelle between snapshots will not be counted by this analysis, so snapshots are recorded every 100,000 time steps to minimize undercounting. At $k_BT/\varepsilon = 1.2$, a typical micelle only loses/gains one polymer per ten snapshots recorded.

Radial distributions of the beads surrounding micelles are also of interest. These are calculated by creating a histogram with bin width dr and then counting the number of beads belonging to a micelle $N_{\mathrm{count}}(r)$ that fall between r and $r+dr$, where r is the distance of the bead from the center of mass of the micelle. Good statistics require averaging this histogram over all micelles in the simulation and over all time steps after the micellar crystal has formed. The average histogram is transformed into a radial density distribution of beads around a micelle by calculating the packing fraction of beads in each bin

$$\phi_i(r) = \frac{\pi}{6} \cdot \frac{\langle N_{\mathrm{count}}(r) \rangle}{4/3\pi \left((r+dr)^3 - r^3 \right) / \sigma^3} \ ,$$

where i refers to either A or B beads. We use $dr = 0.2\sigma$ to balance smooth graphs with the need for long simulation runs to obtain detailed statistics.

5.4 Micellar crystals studied using MD

There are two major challenges faced in using MD to determine equilibrium phases. First, the simulation must last longer than any of the relaxation times in the system. Second, the simulation box size must be chosen properly to avoid finite size effects. The first problem is related to the kinetic temperature at which the system is run. The second problem can become particularly severe for simulating crystals with three dimensional order, where the incorrect choice of an even large box size L can force the system into very distorted ordered phases. These two issues are addressed systematically using simulations of the $A_{10}B_7A_{10}$ polymer. It provides a coarse-grained description of one of the most extensively studied Pluronics, F127 [46]. The conclusions of this study lead to a general methodology valid for any other polymer. For a related approach to this problem, see also Ref. [80].

5.4.1 Micelle crystallization and kinetic temperature

Initial simulations are performed with a system size of $N_{\text{poly}} = 500$ at a kinetic temperature $\frac{k_BT}{\varepsilon} = 1.0$ and with concentrations ϕ_P of 0.05, 0.10, 0.15, 0.20 and 0.25 run over 30 million time steps each. In all cases, the initially randomly placed $A_{10}B_7A_{10}$ polymers aggregate into micelles quickly in a few thousand time steps.

Visual examination indicates that concentrations at and above $\phi_P = 0.15$ are strong candidates for micellar crystals. Micelles are locked in place and only move a few σ from their average positions. The analysis of the order parameter $S(\vec{q})$, however, indicates no long-range ordered structures exist in any of these initial simulations. An inspection of the polymer transfer shows that it remains negligible throughout all simulations. This is confirmed by calculating the mean squared displacement $\langle(\vec{r}(t) - \vec{r}(0))^2\rangle$ averaged over all beads in the system. Figure 5.1 clearly shows non-diffusive behavior at $k_BT/\varepsilon = 1.0$. These results suggest that the individual micelles are quickly frozen in a configuration that is not representative of equilibrium, thus preventing the entire system from reaching thermal equilibrium.

The lack of equilibration of micellar degrees of freedom suggests subsequent runs at a larger kinetic temperature $k_BT/\varepsilon = 1.2$. A single simulation run at $\phi_P = 0.20$ formed a textbook

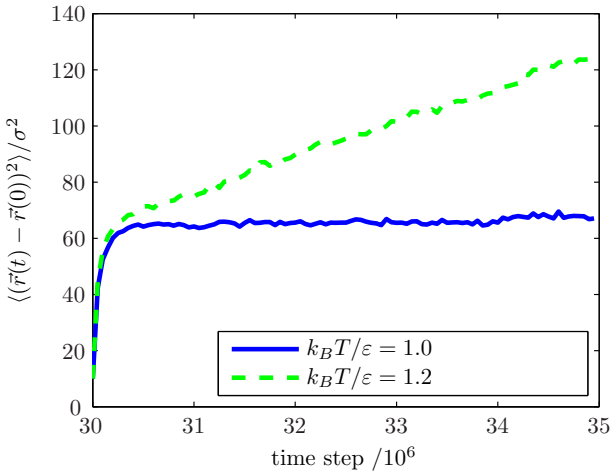

Figure 5.1 (Color online) Mean squared displacement averaged over all beads in the simulation box. This data was obtained from the initial simulation runs of the $A_{10}B_7A_{10}$ polymer at $N_{\text{poly}} = 500$ and $\phi_P = 0.20$. The origin of the time axis is 30 million time steps which indicates that the recording of these results began after the system reached equilibrium. In the case of $k_BT/\varepsilon = 1.0$ this equilibrium is a metastable state, and the beads are not diffusing. The simulation run performed at $k_BT/\varepsilon = 1.2$ formed a fcc lattice around time step 10 million which persisted until the end of the run at 35 million, and the mean squared displacement shows a characteristic diffusive behavior.

fcc lattice after about 10 million steps, and is shown in Figure 5.2. Throughout the duration of the run, the rate of polymer transfer was substantial, about 7% of the polymers in the box every million time steps, *even after* equilibrium is reached. Playing a movie of the simulation shows that after the lattice formed, micelles do not appear to move except by vibrating about their average positions. However, while the micelles appear static, polymers are constantly being exchanged among the micelles, so that any individual polymer will eventually explore the entire simulation box. This is independently confirmed by the analysis of the mean squared displacement of beads in Figure 5.1, which shows a classic diffusion result at $k_BT/\varepsilon = 1.2$.

The behavior of the number of micelles N_{mic} in the box as a function of simulation time is of particular interest. In simulation runs where micellar crystals are found, the system reaches

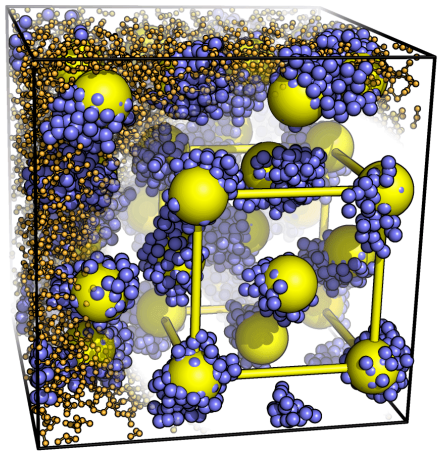

Figure 5.2 (Color online) Snapshot of a $A_{10}B_7A_{10}$ polymer simulation run at $\phi_P = 0.20$, $k_BT/\varepsilon = 1.2$, $N_{\text{poly}} = 500$, taken after the fcc lattice formed. A beads are represented by orange spheres, and are shown with a reduced radius so they do not obscure important details. Orange lines indicate bonds between these beads. B beads are shown in blue with a radius of 0.6σ. Large yellow spheres are placed on the lattice reconstructed from $S(\vec{q})$. Every yellow sphere is sitting on a micelle, visually confirming a perfect fcc crystal. The A beads have been removed around a single unit cell of the lattice and yellow lines added to guide the eye. All snapshots are generated using PyMol [56].

a plateau where N_{mic} remains constant, see Figure 5.3 for an example. Despite their dynamic character even at equilibrium, N_{mic} then remains constant for the duration of the run. This correlation is a general feature in all simulation runs performed. *Every single* one that leads to a stable plateau in N_{mic} as a function of time formed a micellar crystal confirmed by peaks in the order parameter $S(\vec{q})$.

5.4.1.1 Box size effects

The influence of the simulation box size choice is assessed by running additional simulations with $N_{\text{poly}} = 500$, 600, 800, 900, and 1000 with fixed concentration $\phi_P = 0.20$ and $\frac{k_BT}{\varepsilon} = 1.2$. Several different random initial configurations are used at each system size to ensure repeatability. Table 5.1 summarizes the results of all these simulation runs.

No. configs	N_{poly}	N_{mic}	Ordering	$\langle N_{\text{agg}} \rangle$
1	500	32	*textbook* fcc	15.625
3	500	unstable	none	
1	600	36	none	16.67
3	600	36	distorted bcc	16.67
5	800	48	distorted bcc	16.67
5	900	54	*textbook* bcc	16.67
1	900	55	distorted bcc	16.36
2	1000	unstable	none	
4	1000	60	distorted bcc	16.67

Table 5.1 Summary of results obtained from initial test runs.

No. configs	Targeted	N_{poly}	N_{mic}	Ordering
1	16 micelle bcc	267	16	*textbook* bcc
4	108 micelle fcc	1800	unstable	none
4	128 micelle bcc	2134	128	*textbook* bcc
3	250 micelle bcc	4168	unstable	none

Table 5.2 Summary of simulation results from testing the algorithm on $A_{10}B_7A_{10}$.

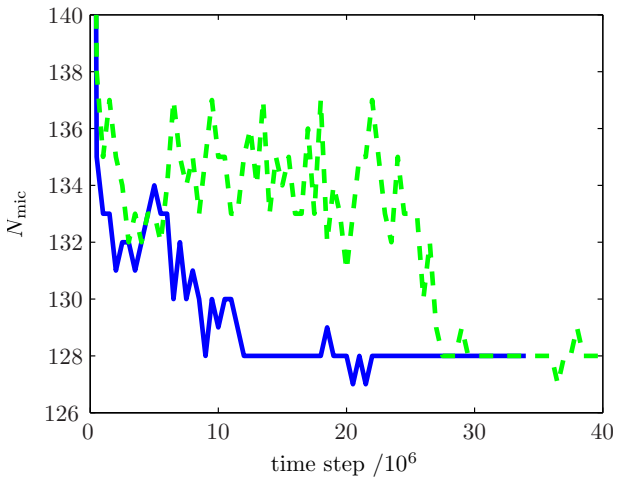

Figure 5.3 (Color online) Examples of the behavior of N_{mic} during a simulation run of the $A_{10}B_7A_{10}$ polymer. Results are shown here for two independent runs with $\phi_P = 0.20$, $k_BT/\varepsilon = 1.2$ and $N_{\mathrm{poly}} = 2134$. The origin of the time axis (t=0) indicates the time step where the entire simulation started from a random initial configuration. Note how both systems eventually plateau at the same state $N_{\mathrm{mic}} = 128$, though it occurs at different times.

Interestingly, the additional three simulation runs at $N_{\mathrm{poly}} = 500$ do not exhibit fcc structures. Instead, each of them remains unstable with N_{mic} never achieving a constant plateau and $S(\vec{q})$ is devoid of peaks. Larger system sizes do reach stable ordered structures with constant N_{mic} and numerous peaks in $S(\vec{q})$. Most of the structures that occur appear bcc when examined visually, but a detailed analysis of the lattices indicates that many of them are distorted. One obviously distorted structure is depicted in Figure 5.4 where the lattice is body centered tetragonal with $c/a = 1.5$.

Other distortions include body centered tetragonal with $c/a = 1.054$ for $N_{\mathrm{poly}} = 800$ and a lattice that appears to be almost exactly bcc when $N_{\mathrm{poly}} = 1000$, except that the central micelle in the unit cell is shifted slightly from the true center. Lastly, a textbook bcc lattice is formed in the simulation runs with $N_{\mathrm{poly}} = 900$.

Examining the average aggregation number leads to a very illuminating result. It is found that for almost all simulations these numbers are *identical*. This indicates that the average

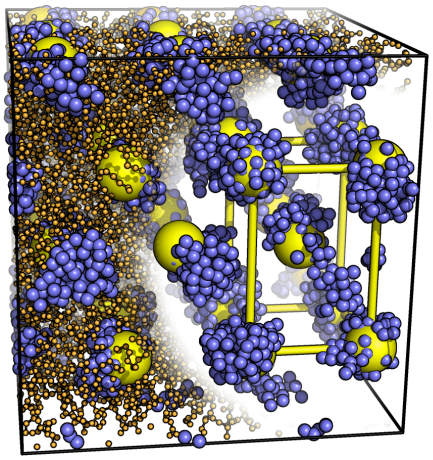

Figure 5.4 (Color online) Snapshot of a $A_{10}B_7A_{10}$ polymer simulation run at $\phi_P = 0.20$, $k_BT/\varepsilon = 1.2$, and $N_{\text{poly}} = 600$ taken after the lattice formed. The resulting lattice in this system is a body-centered tetragonal with $c/a = 1.5$. Coloring conventions are identical to Figure 5.2

.

micelle aggregation number is independent of the box length (at a fixed ϕ_P) and is only a function of the polymer structure, kinetic temperature and concentration.

Assuming without proof that either the fcc or the bcc lattice represents the real thermodynamic equilibrium of the system, the previous observation then suggests a way to generate *magic numbers* of polymers to simulate equilibrium states free of finite size effects. In a bcc lattice, each cubic unit cell contains $C = 2$ micelles, while the fcc lattice contains $C = 4$. Therefore, in order to obtain a bcc or fcc lattice with M by M by M unit cells in a cubic simulation box, the number of polymers needed to achieve this is given by

$$N_{\text{poly}} = CM^3\langle N_{\text{agg}}\rangle. \tag{5.5}$$

5.4.2 MD Simulations without finite size effects

An algorithm to simulate micellar crystals without finite size effects follows very naturally from the previous results, and is summarized in the following steps.

1. *Concentration selection:* The concentration must be chosen high enough so that micelles pack into a potential crystalline state.

2. *Temperature selection:* The temperature should be chosen large enough to ensure a significant rate of polymer transfer, but low enough that the micellar crystals are not in a disordered phase. As a rule of thumb, we have been using a polymer transfer around 10% of the polymers in the box every million steps.

3. *System size selection:* Calculate the average micelle number via test simulations and use Equation 5.5 to determine the final system sizes to perform simulations on.

4. *Ensure reproducibility:* The formation of micellar crystals is a stochastic process so several simulations with different initial configurations must be run.

The advantage of this algorithm is that steps 1-4 can be accomplished with relatively modest computer resources on small system sizes, leaving the production runs with large polymer numbers as the only computationally intensive calculations.

5.4.3 Micellar crystals of general $A_nB_mA_n$ triblocks

5.4.3.1 $A_{10}B_7A_{10}$

Further simulation runs are carried out on the $A_{10}B_7A_{10}$ polymer system to test the algorithm. These additional simulations target 16, 128 and 250 micelle bcc configurations, along with 32 and 108 micelle fcc ones. The simulations are summarized in Table 5.2. None of the simulation runs targeting fcc phases ever reach a stable number of micelles and correspondingly, $S(\vec{q})$ indicates there is no long-range order. On the other hand, both the 16 and 128 micelle bcc configurations are perfect and completely reproducible with the expected ordering confirmed by the structure factor. Figure 5.5 shows a snapshot of the 128 micelle bcc phase after it forms. Further discussion on the calculated structure factors are presented below. Larger simulations attempting the formation of a 250 micelle bcc crystal never resulted in a stable N_{mic} plateau.

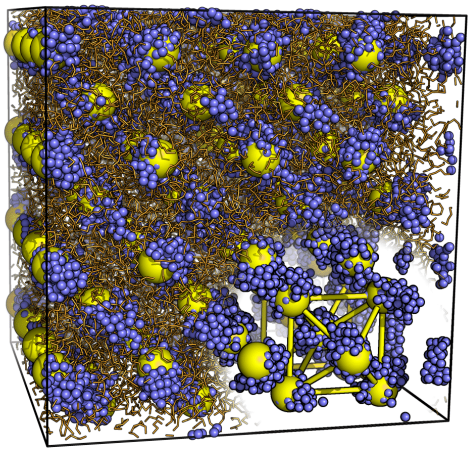

Figure 5.5 (Color online) Snapshot of a $A_{10}B_7A_{10}$ simulation run at $\phi_P = 0.20$, $k_BT/\varepsilon = 1.2$, and $N_{\text{poly}} = 2134$, taken after the bcc lattice formed. Coloring conventions are identical to Figure 5.2

5.4.3.2 $A_{20}B_{14}A_{20}$

The $A_{20}B_{14}A_{20}$ polymer is also studied as it provides a closer representation to the real Pluronic F127, since there are twice as many monomers in a polymer and the level of coarse-graining is less, as discussed in Ref. [75]. Applying the algorithm developed above to search for micellar crystals, the concentration was selected at $\phi_P = 0.20$ and the kinetic temperature at $k_BT/\varepsilon = 1.9$. Initial simulation runs establish that $\langle N_{\text{agg}} \rangle = 22.5$. The results, summarized in Table 5.3, are similar to those for the smaller $A_{10}B_7A_{10}$ polymer, with the bcc structure being the most commonly found. The fcc phase again only appears in a single simulation run and is not reproducible.

5.4.3.3 $A_6B_7A_6$

Given the prevalence of bcc lattices so far, a polymer with shorter A-blocks ($A_6B_7A_6$), expected to form more *crew-cut* micelles and hence more prone to assemble into a fcc lattice, is also investigated. Applying the algorithm developed above, the concentration is selected at $\phi_P = 0.15$, the kinetic temperature at $k_BT/\varepsilon = 1.1$, and the average aggregation number

No. configs	Targeted	N_{poly}	N_{mic}	Ordering
2	16 micelle bcc	360	16	*textbook* bcc
1	32 micelle fcc	720	32	*textbook* fcc
1	32 micelle fcc	720	35	distorted bcc
2	32 micelle fcc	720	unstable	none
1	54 micelle bcc	1215	54	*textbook* bcc
3	54 micelle bcc	1215	55	distorted bcc
2	108 micelle fcc	2430	unstable	none

Table 5.3 Summary of simulation results from testing the algorithm on $A_{20}B_{14}A_{20}$ at $k_BT/\varepsilon = 1.9$.

No. configs	Targeted	N_{poly}	N_{mic}	Ordering
5	16 micelle bcc	222	16	*textbook* bcc
5	32 micelle fcc	444	unstable	none
5	54 micelle bcc	750	54	*textbook* bcc
5	108 micelle fcc	1500	unstable	none
3	128 micelle bcc	1778	128	*textbook* bcc
1	128 micelle bcc	1778	136	distorted bcc
3	250 micelle bcc	3472	unstable	none
4	432 micelle bcc	6035	unstable	none

Table 5.4 Summary of simulation results from testing the algorithm on $A_6B_7A_6$ at $k_BT/\varepsilon = 1.1$.

is found to be $\langle N_{\text{agg}} \rangle = 13.875$. Again, the bcc micellar crystal is most commonly found, as shown in the results in Table 5.4. No fcc phases are found at this kinetic temperature.

However, the rate of polymer transfer at $k_BT/\varepsilon = 1.1$ is larger than 10% per million steps, so some additional simulation runs are performed at a slightly reduced temperature $k_BT/\varepsilon = 1.07$, where polymer transfer is slightly reduced. In this case a completely reproducible fcc structure is found for relatively small system sizes, but no fcc lattices could be stabilized for larger ones as shown in the results in Table 5.5.

No. configs	Targeted	N_{poly}	N_{mic}	Ordering
4	32 micelle fcc	444	32	*textbook* fcc
1	32 micelle fcc	444	unstable	none
2	108 micelle fcc	1500	unstable	none
2	108 micelle fcc	1500	103	distorted bcc
1	108 micelle fcc	1500	104	distorted bcc

Table 5.5 Summary of simulation results from testing the algorithm on $A_6B_7A_6$ after cooling to $k_BT/\varepsilon = 1.07$.

5.4.4 Non-cubic morphologies

It is well known that block copolymer systems form ordered phases other than those with cubic symmetry [33]. Both lamellar planes (1D) and hexagonal (2D) ordering of cylindrical micelles are commonly found. With the lower dimensionality, the influence of finite size effects in distorting the phases is reduced because these systems rotate to fit in the periodic simulation box. Consequently, hexagonal and lamellar phases are are much simpler to obtain using MD simulations than cubic phases.

For instance, the $A_6B_7A_6$ polymer forms a hex phase at a packing fraction of $\phi_P = 0.25$ every time with no need for multiple runs from different initial conditions. No hex or lamellar phases are found for $A_{10}B_7A_{10}$ or $A_{20}B_{14}A_{20}$, consistent with experiments for Pluronic F127 [46]. $A_4B_9A_4$, not otherwise mentioned in this paper, exhibits all three phases as concentration is increased: cubic, hexagonal, and lamellar, consistent with experiments for Pluronic P65 [46].

Given that hexagonal and lamellar phases are relatively simple to obtain, we focus our analysis on micellar crystals with periodicity in three dimensions.

5.5 Dynamics of crystal formation

Molecular dynamics simulations not only allow the prediction of equilibrium phase diagram, but also describe the dynamics of micelles as they evolve towards thermal equilibrium. In the simulations presented previously, micelles quickly form from a completely random configuration

of polymers, and then after a long simulation time (10 to 20 million time steps) order into a micellar crystal. We now present a quantitative picture of the dynamics of the polymers and micelles as the system approaches equilibrium. All results in this section are obtained from the analysis of the simulations of the $A_{10}B_7A_{10}$ polymer.

5.5.1 Polymer transfer is an activated process

Polymer transfer plays an important role in achieving the formation of micellar crystals in the simulations discussed above. If there is too little, the single micelle degrees of freedom do not reach equilibrium. Too much pushes the system into a disordered state. Moreover, the rate of polymer transfer is extremely sensitive to the kinetic temperature. An equilibrated $N_{\text{poly}} = 2134$ bcc micellar crystal was taken as an initial configuration for additional simulations that continued with k_BT/ε ranging from 1.0 to 1.3. Figure 5.6 shows the results. The rate of polymer transfer r_{PT} starts near 0 at $k_BT/\varepsilon = 1.0$ and increases exponentially, following an Arrhenius form

$$r_{\text{PT}} = r_O \exp(-\frac{\Delta G^\sharp}{k_BT}) \ . \tag{5.6}$$

That is to say that polymer transfer is an activated process. Curve fitting, we find

$$\Delta G^\sharp \approx 10\varepsilon \ . \tag{5.7}$$

Figure 5.6 also includes results from simulations performed using a Langevin thermostat [81]. It controls the temperature by adding an additional force to every particle $\vec{F} = -\gamma\vec{v} + \vec{F}_{\text{rand}}$ where the magnitude of the random force $\vec{F}_{\text{rand}}$ and γ set the temperature through the fluctuation dissipation theorem [81]. The results for two different values of γ, which are plotted in Figure 5.6, show the dependence of the polymer transfer for different drag coefficients.

In Figure 5.6, the slope of the lines in the inset plot are universal for both thermostats and both values of γ. The universality of this value implies that the calculated $\Delta G^\sharp$ is the free energy barrier between a polymer in a micelle to a transition state between micelles. While their slopes are universal, the y-intercepts (related to r_O) should depend on the diffusion coefficient

of the hydrophilic beads, which, from the Einstein relation [82] is inversely proportional to the drag coefficient. This is reflected in the offsets of the various plots in Figure 5.6, which are clearly different.

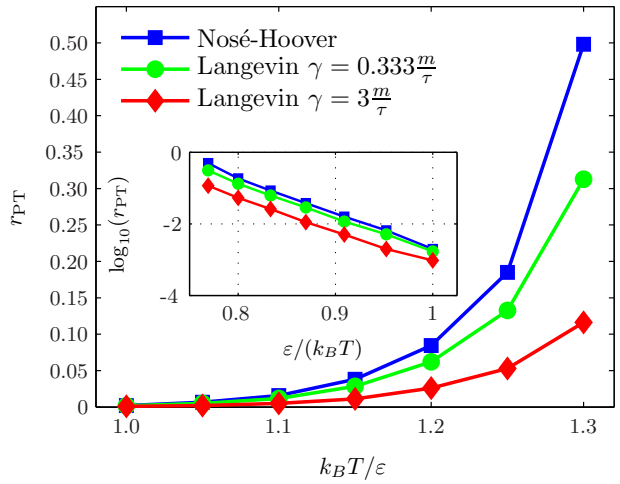

Figure 5.6 (Color online) Polymer transfer r_{PT} versus temperature calculated from simulation runs of the $A_{10}B_7A_{10}$ polymer at $\phi_P = 0.20$ and $N_{\mathrm{poly}} = 2134$. All simulations start from an already equilibrated bcc phase. Results are included for the Nosé-Hoover thermostat and Langevin thermostat with two different values of γ. The inset plots the same data as a plot of $\log_{10}(r_{\mathrm{PT}})$ versus $\varepsilon/(k_BT)$ to show that the slopes of the resulting lines $(-\Delta G^\sharp)$ are universal.

The difference in free energy between a polymer within a micelle and in the transition state entirely surrounded by solvent and hydrophilic beads can be roughly estimated as $\Delta G^\sharp \sim m_{exp} \cdot \varepsilon$, where m_{exp} is the number of hydrophobic beads exposed to solvent. If the length of hydrophobic block is low, $m_{exp} \sim m$. Thus for the $A_{10}B_7A_{10}$ polymer analyzed here, it is expected that $\Delta G^\sharp \sim 7\varepsilon$, which is consistent with the measured value. Hydrophobic beads in polymers with a longer hydrophobic block are expected to form a globule in the transition state, leading to a slower increase of $\Delta G^\sharp$ going as $m_{exp} \sim m^{2/3}$. Assuming that r_O remains constant as m increases, then maintaining the same rate of polymer transfer r_{PT} will require increasing the kinetic temperature by the same factor. These considerations agree remarkably

with the kinetic temperature of $k_BT/\varepsilon = 1.9$ selected for the $A_{20}B_{14}A_{20}$ polymer simulations,

$$\left(\frac{m_{\text{new}}}{m_{\text{prev}}}\right)^{2/3} \cdot k_BT_{\text{prev}}/\varepsilon = \left(\frac{14}{7}\right)^{2/3} \cdot 1.2 = 1.9$$

.

However, as the size of the hydrophobic block grows, the transition state should also have an additional contribution due to the free energy cost of passing a globule through the brush formed by the hydrophilic coronas. So this simple estimate should eventually break down.

Implicit in our arguments is that polymer transfer accounts for the diffusive behavior in Figure 5.1. Therefore, the diffusion coefficient can be roughly estimated from the time τ_{PT} it takes a polymer to travel the nearest neighbor micelle distance a_L. Comparing this estimate with the fit to Figure 5.1, a good agreement is obtained:

$$\begin{aligned} \frac{\langle(\vec{r}(t)-\vec{r}(0))^2\rangle}{t} &= 1.3\cdot 10^{-5}\frac{\sigma^2}{\Delta t} \\ \frac{a_L^2}{\tau_{PT}} = a_L^2 r_{PT} &= 1.1\cdot 10^{-5}\frac{\sigma^2}{\Delta t}\ . \end{aligned} \tag{5.8}$$

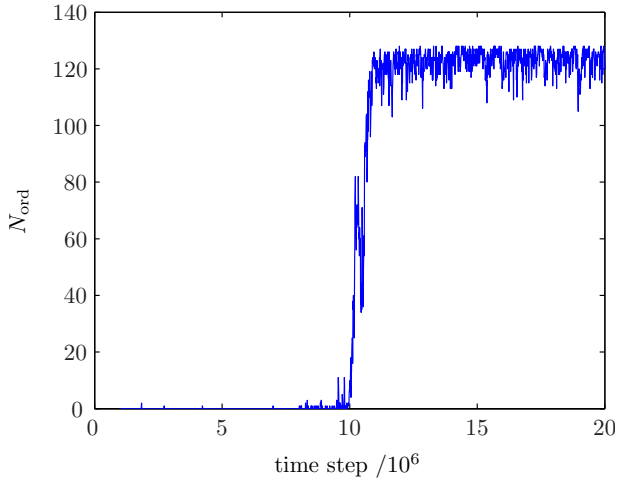

Figure 5.7 (Color online) Number of micelles in the ordered phase N_{ord} as a function of time for a single simulation run of the $A_{10}B_7A_{10}$ polymer at $\phi_P = 0.20$ and temperature $k_BT/\varepsilon = 1.2$. The origin of the time axis (t=0) on this plot indicates the time step where the simulation was started from a random configuration.

5.5.2 Dynamics of micellar crystal formation

The structure factor $S(\vec{q})$ is sufficient as an order parameter to determine if the entire system is in an ordered state, but it reveals no information about the dynamics before that final state is formed. For this, we turn to the bond order analysis [83] and apply it to all the micelles at every time step in simulation runs performed on the $A_{10}B_7A_{10}$ polymer with $N_{\text{poly}} = 2134$. In short, at any given time step where a micellar crystal may have partially formed, the bond order analysis identifies those micelles belonging to the ordered and the disordered phases.

One simple way to examine the results is to count the number of ordered micelles N_{ord} in the simulation box at every time step. An example from one simulation run is shown in Figure 5.7, plots for all other simulation runs performed are qualitatively very similar, though the ordered phase appears at different times.

After starting the simulation shown in Figure 5.7 from a random configuration, micelles quickly form in only a few thousand time steps and single micelle degrees of freedom are equilibrated before one million time steps have passed. The system then explores configuration space as polymers are transferred and micelles travel anywhere from a few σ up to 100σ without any micelles appearing ordered until time step 10 million. At this point, the number of ordered micelles grows very quickly over the next one million steps until all 128 micelles are in the bcc lattice and remain there for the duration of the simulation.

During this short time span of N_{ord} growth, only 7% of the polymers in the box are transferred between micelles, such a small amount that it cannot fully account for the ordering of all the micelles. During the same time interval, a detailed analysis shows that some micelles move significantly (up to 10σ) before the ordered phase finishes forming. This may suggest that micellar crystal formation is a two step process, where first individual micelles are equilibrated by polymer transfer followed by a second step where polymer transfer becomes irrelevant and the actual crystal grows via the movement of micelles.

However, some additional simulations performed to test this hypothesis do not support it. Single micelle degrees of freedom in these tests are first equilibrated at $k_BT/\varepsilon = 1.2$ for 5

million time steps and then the kinetic temperature is quenched to $k_BT/\varepsilon = 1.0$ to significantly reduce polymer transfer and allow micelle movement. None of these simulation runs resulted in the formation of an ordered phase. We therefore conclude that a significant amount of polymer transfer remains a critical component in the actual growth of ordered micellar crystals.

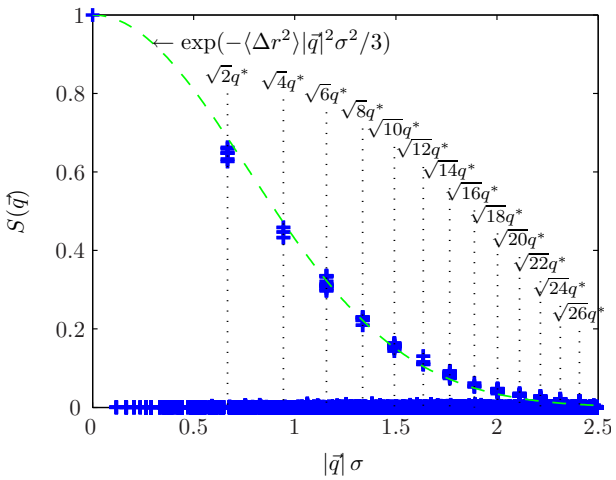

Figure 5.8 (Color online) Structure factor calculated after the lattice formed for the $A_{10}B_7A_{10}$ polymer simulation run at $\phi_P = 0.20$, $k_BT/\varepsilon = 1.2$, and $N_{\text{poly}} = 2134$. The full 3D $S(\vec{q})$ is plotted as a scatter plot of $S(\vec{q})$ versus $|\vec{q}|$. The multiplicity of the various peaks can be seen. Vertical dotted lines indicate the location of identified peaks, and their positions relative to $q^* = 4 \cdot 2\pi/L$ are also noted (the factor of 4 is included because there are 4 unit cells along the box length L).

5.6 Properties of micellar crystals

5.6.1 Structure factor

We calculate the structure factor (Equation 7.7) for the equilibrium state of every simulation run performed and use it as an order parameter to determine if a micellar crystal is present. In those systems where the crystal does form it is a perfect single crystal and, correspondingly, a large number of peaks can be identified in $S(\vec{q})$ as shown in Figure 5.8. Peaks occur in reciprocal space at discrete points $\vec{q} = \vec{G}$, where $\vec{G}$ are the reciprocal vectors of the corresponding lattice.

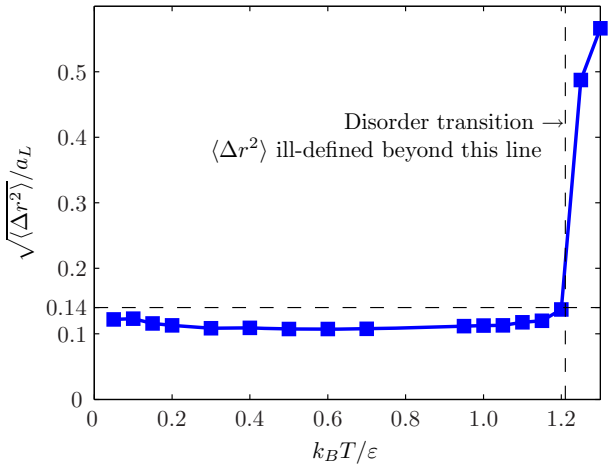

Figure 5.9 (Color online) Fluctuations in micelle positions vs. temperature calculated for a $A_{10}B_7A_{10}$ polymer simulation run at $\phi_P = 0.20$, $k_BT/\varepsilon = 1.2$, and $N_{\text{poly}} = 900$. The micellar crystal was formed at $k_BT/\varepsilon = 1.2$ and then cooled to the target temperature without disrupting the lattice. Simulation runs heated to a higher temperature do disrupt the lattice and Δr^2 becomes ill-defined. Fluctuations on the y-axis are plotted as a ratio relative to a_L, the nearest neighbor distance in the lattice.

Peaks evident in Figure 5.8 decrease in magnitude for larger values of $\vec{G}$. This damping is expected to follow a Debye-Waller factor [84] approximately described as

$$S(\vec{q} = \vec{G}) \propto \exp(-\langle \Delta r^2 \rangle |\vec{G}|^2/3) \ , \tag{5.9}$$

where $\langle \Delta r^2 \rangle$ is the mean square displacement of the micelle center of mass from its ideal lattice position. A curve fit, shown in Figure 5.8, is in excellent agreement with Equation 5.9. The Lindemann ratio f_L is defined as

$$\sqrt{\langle \Delta r^2 \rangle} = f_L a_L \tag{5.10}$$

where a_L is the nearest neighbor distance between micelles. Using the parameters of the curve fit yields $f_L \approx 0.14$.

The Lindemann parameter f_L can alternatively be computed directly by measuring the mean square displacement of micelles about their average lattice positions. The results, shown in Figure 5.9 agree remarkably well with the estimate from the Debye-Waller factor. Further-

more, it has been established empirically that approximately at $f_L = 0.13$ solids melt into disordered states [85], a result that is also supported from our simulations as we observed no micellar crystals form at kinetic temperatures greater than $k_BT/\varepsilon = 1.2$

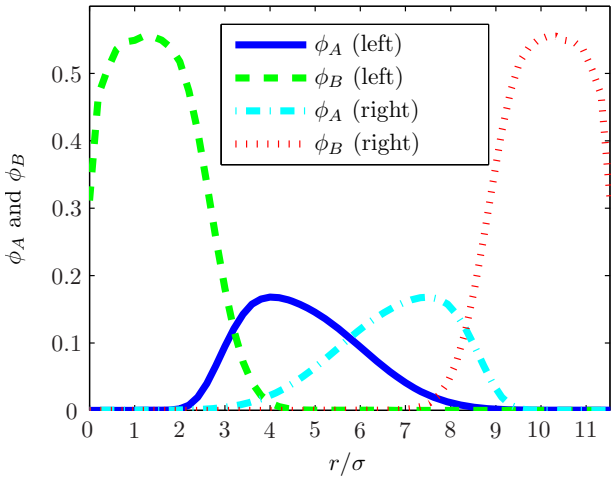

Figure 5.10 (Color online) Radial density distribution for two nearest neighbor micelles superimposed with a separation of the nearest neighbor distance in the bcc lattice. The y-axis plots the volume fraction of the different beads belonging to a micelle in the local environment around the micelle. The results were calculated from a $A_{10}B_7A_{10}$ simulation run at $\phi_P = 0.20$, $k_BT/\varepsilon = 1.2$, $N_{\text{poly}} = 2134$ and averaged over all micelles and time steps after the micellar crystal has formed.

5.6.2 Structure of the lattice of micelles

Micelles in the $A_{10}B_7A_{10}$ polymer system are arranged with the centers of mass sitting on a bcc lattice with a nearest neighbor spacing of $a_L = 11.53\sigma$. This number remarkably agrees with the length of the polymer if stretched completely into a straight line across the diameter of a circle from the center of one nearest neighbor micelle to the opposite one, $N_{\text{mon}} \cdot r_0 = 27 \cdot 0.83\sigma = 22.41\sigma \sim 2a_L$, where $r_0 = 0.83\sigma$ is the equilibrium bond length. This suggests that polymers are maximally stretched, either along the diameter or in a bent configuration. Detailed visual examinations of simulation snapshots indicate that there are a

significant number of polymers in the bent configuration, but there are also some in the linear extended configuration. The micellar cores are liquid-like, and over time, a given polymer constantly switches from linear to bent configurations.

The previous arguments also suggest a significant amount of overlap of the coronas between two neighboring micelles in the lattice. To examine this more quantitatively we calculate the micelle density distribution as a function of the radius averaged over all micelles and all time steps in the simulation after the micellar crystal has formed. Figure 5.10 shows the results, confirming the overlap. Hydrophilic A beads from one micelle explore the solvent until occasionally bumping into the hydrophobic core of one of the nearest neighbor micelles.

A remarkable aspect of micellar crystals found in this work is the stability of the average aggregation number. If the polymers within the micelle are maximally stretched, the core of the micellar radius is $R_c = mr_0\sigma/2$, so the aggregation number can be estimated from

$$\langle N_{agg} \rangle = \frac{4\pi(mr_0/2)^3\sigma^3}{mr_0\sigma^3} = \frac{\pi}{6}m^2r_0^2 \ . \tag{5.11}$$

For the $A_{10}B_7A_{10}$ this yields $\langle N_{agg} \rangle = 17.7$, in good agreement with the simulation results in Table 5.1. Aggregation numbers for the other polymers simulated do not agree, implying that the polymers in those systems are not maximally stretched.

5.7 Conclusions

5.7.1 Summary of results

Cubic micellar crystals of $A_nB_mA_n$ polymers form in MD simulations at sufficiently high concentrations. In order to form the crystals, high enough kinetic temperatures are needed to enable polymer transfer between micelles, which is critical for equilibrating the system. The polymer transfer process is activated and described by transition state theory. It results in an apparent diffusive behavior of the individual polymers while the lattice of micelles remains stable. Excessive polymer transfer at even higher kinetic temperatures triggers a disorder phase transition to a micelle liquid.

In the process of forming the ordered phase, the system spends a long time in a micellar liquid phase equilibration period where no ordered nucleates are present. Eventually, a large nucleation event takes place and the micellar crystal then grows very quickly, filling the entire simulation box in a relatively short time span. During this growth period, polymer transfer and movement of micelles are both crucial in the formation of the final micellar crystal. The preferred ordering near the disordered transition for all triblocks studied in this system is the bcc lattice. Only at lower kinetic temperatures and for polymers with short hydrophilic groups ($A_6B_7A_6$ at $k_BT/\varepsilon = 1.07$) is there some evidence for a stable fcc phase. These results are summarized in Figure 5.11.

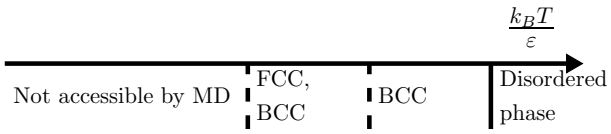

Figure 5.11 Summary of the phase diagram encompassing all simulated triblocks $A_nB_mA_n$ (all forming cubic phases with long range order). Near the disorder transition, bcc is always favored and fcc lattices only begin to appear at lower temperatures. At even lower temperatures polymer transfer becomes negligible and MD would require prohibitively long simulations to reach equilibrium.

All micellar crystals obtained in this work are perfect single crystals displaying a high degree of order. Even in a periodic simulation box with four unit cells along a side, a single additional micelle can disrupt the resulting lattice significantly. The number of micelles is controlled by changing the number of polymers in the box, as the system displays a remarkable stability in the average aggregation number of the micelles that form. In the equilibrium lattice, micelles are closely packed with a significant amount of overlap between the coronas of neighboring micelles. Polymers in the micelles are highly stretched across the liquid hydrophobic core through the solvent and qualitatively well described within the strongly stretched approximation [77, 86].

5.7.2 Stability of the bcc lattice near the disorder transition

Our simulations results show a strong preference for bcc lattices near the disorder transition. A similar result has been experimentally observed for PS-PI diblocks [70], where it has been attributed to the fact that near the disordered phase, micelle aggregation numbers are small. The phase diagram of f-star polymer systems shows that fcc lattices are only stable for large number of arms $f > 60$, while bcc lattices are favored when the number of arms is small [87]. By considering polymeric micelles as f-star polymers, where the number of arms f is given by $f \sim 2\langle N_{agg}\rangle$ then bcc lattices are favored when the aggregation numbers are small $\langle N_{agg}\rangle \lesssim 30$. This argument, however, hinges on two key assumptions: the dynamic nature of micelles does not play a significant role and that the hydrophilic blocks are sufficiently long. The first assumption is already somewhat problematic given the importance of polymer transfer found in this work. As for the second assumption, a criteria establishing how long hydrophilic blocks should be has been put forward in Ref. [73], where it is shown that if the size of the corona is L_c and the core radius R_c, bcc lattices are favored for $L_c/R_c > 1.5$. Our results for the $A_{10}B_7A_{10}$ and $A_6B_7A_6$ yield $L_c/R_c \sim 0.9$ and $L_c/R_c \sim 0.7$. It is therefore not possible to attribute the stability of the bcc lattices observed in our simulations as being a consequence of the small aggregation numbers $\langle N_{agg}\rangle \lesssim 30$.

It is tantalizing to interpret the stability of the bcc lattice in terms of the Alexander-McTague (AM) scenario [88], where it was argued that bcc should be generally expected to be the stable phase near a (weakly first order) disorder transition. Subsequent analysis however, showed that cubic lattices other than bcc cannot be ruled out near the disorder transition [89, 90]. In Ref. [89, 90] it is shown that the characteristic property of bcc lattices is that their free energy is closer to the disordered state, thus suggesting that bcc lattices follow an Ostwald step rule [91], where the solid phase that nucleates first is the one whose free energy is closest to the disordered (or fluid) state. In this case, the complete crystallization process would require an additional step, where after the bcc crystallites are formed, they gradually evolve towards the stable thermodynamic phase. Certainly, it follows from our results that fcc lattices are difficult to obtain by MD for the systems we simulate, but at least in one system ($A_6B_7A_6$

at $k_BT/\varepsilon = 1.07$) the fcc lattice has been obtained reproducibly, and did not proceed through an intermediate bcc step. In addition, for this very same system, closer to the disordered state ($k_BT/\varepsilon = 1.1$), no fcc structure was found to be stable. Although not completely conclusive, our results are more consistent with the bcc as being a stable thermodynamic phase near the disordered phase.

There are serious limitations in identifying micelles as simple particles, because as the disordered phase is approached micelle aggregation numbers decrease and polymers become essentially free. So the disordered phase is not a simple liquid as it is assumed by AM and all subsequent work. Similar analysis in polymer melts [92], shows that in the vicinity of the spinodal, the only possible phase with cubic symmetry is bcc. Beyond the spinodal, other structures are favored [93]. Based on the previous discussion, we attribute the stability of the bcc phases observed in our simulations as a reflection of the admittedly non-rigorous statement that bcc phases are *usually favored* near the disordered transition. We defer to future work to establish this result within a rigorous framework, where all the nuances involved in micelle formation are properly taken into account.

5.7.3 Implications for Pluronic systems

The $A_{10}B_7A_{10}$ and $A_{20}B_{14}A_{20}$ polymers discussed in this paper provide coarse-grained descriptions of Pluronic F127. All simulations have been carried out in very good solvent conditions for the A beads and at total volume fractions of 15–20%. Experimental results in this region of the phase diagram are surprisingly disparate. Simple cubic [94], bcc [95] and fcc [96] have all been proposed as the structure in this region. Very recently, on the basis of new experimental data, the situation has been thoroughly reviewed by Li et al. [97] (although for significantly higher temperatures), see also Ref. [98], but without clear conclusive results. Our theoretical analysis clearly favors the bcc lattice close to the disorder transition. The comparison of our results with the experimental F127 system is more accurate at low temperatures (at 20-25 C), where the water can be considered as a good solvent for PEO, as discussed in Ref. [24].

The phase diagram of other Pluronic systems, such as P65 have also been investigated and compare very well with experimental results, as briefly mentioned previously.

5.7.4 Outlook

We have shown that MD allows a detailed investigation of both the dynamics as well as the thermodynamic equilibrium of micellar crystals. Many studies have been performed by modeling micelles as point particles, where the complex structure of the micelles is accommodated through refined two-body potentials, either derived analytically or empirically. While successful in many situations, two-body potentials do not account for the dynamic nature of micelles, which play a critical role in determining the phases of the system, particularly near the disorder transition.

There are a few areas where further work needs to be performed. First, all simulations in this work have been performed in good solvent conditions. MD with implicit solvents of different quality are also of great interest, especially to determine the phases of Pluronic systems over a wide range of temperatures [24]. Next, the largest micellar crystal formed in this work contains 2134 polymers (57,618 beads). We were unable to find any order in larger systems, even after running as many as 50 million time steps. It is possible that significantly longer simulations may be required for larger systems. Also, the range of applicability of MD is restricted to a relatively narrow range near the disorder transition, as schematically shown in Figure 5.11. Finally, the role of the solvent will need to be investigated in more detail, as it is found in the Rouse vs. Zimm dynamics for simple homopolymers [82]. Future studies will be necessary to completely clarify and expand all these issues.

The relevance of our study goes beyond pure systems. In Ref. [99] for example, Pluronic polymers have been used to template the growth of an inorganic phase of calcium phosphate, aimed at creating new polymer nanocomposites with lightweight/high strength properties or that mimic the structure of real bone. An understanding of the pure systems is clearly a prerequisite for accurate models of polymer nanocomposites. We hope to report more on this topic in the near future.

5.8 Acknowledgments

We acknowledge interest and discussions with S. Mallapragada, K. Schmidt-Rohr and G. Grason. We also thank J. Schmalian for clarifications regarding the Alexander-McTague paper. This work is supported by DOE-BES through the Ames lab under contract no. DE-AC02-07CH11358 and partially supported by the NSF through grant DMR-0426597.

CHAPTER 6. GENERAL PURPOSE MOLECULAR DYNAMICS SIMULATIONS FULLY IMPLEMENTED ON GRAPHICS PROCESSING UNITS

A paper published in the *Journal of Computational Physics* [3]

Joshua A. Anderson, Chris D. Lorenz, and Alex Travesset

6.1 Abstract

Graphic Processing Units (GPUs), originally developed for rendering real-time effects in computer games, now provide unprecedented computational power for scientific applications. In this paper, we develop a general purpose molecular dynamics code that runs entirely on a single GPU. It is shown that our GPU implementation provides a performance equivalent to that of fast thirty processor core distributed memory cluster. Our results show that GPUs already provide an inexpensive alternative to such clusters and discuss implications for the future.

6.2 Introduction

Fuelled by the dramatic increases in computing power over the years, the impact of computational methods in the traditional sciences has been gigantic. Yet, the quest for new technologies that enable faster and cheaper calculations is more fervent than ever, as there are a vast number of problems that are on the brink of being solved if only a relatively modest increase in computer power were available.

Molecular dynamics (MD) has emerged as one of the most powerful computational tools [81], as it is capable of simulating a huge variety of systems both in and out of thermodynamic

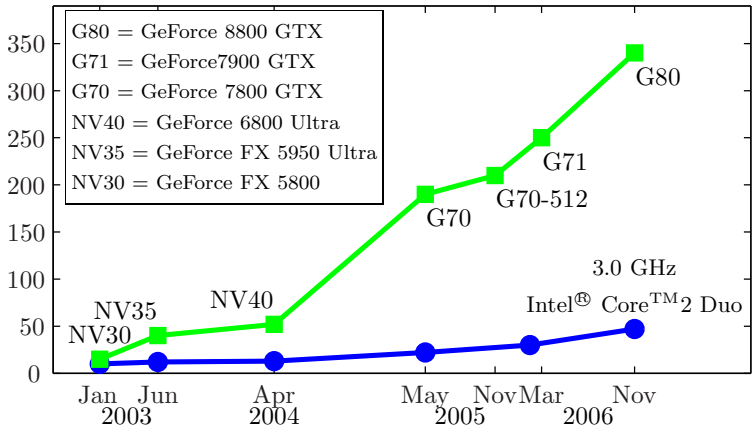

Figure 6.1 Performance of CPUs (blue circles) and GPUs (green squares) over the last few years. Figure courtesy of NVIDIA, and adapted from Ref. [100].

equilibrium. During the last decade, general purpose MD codes such as LAMMPS [22], DLPOLY [52], GROMACS [101], NAMD [23], and ESPResSO [102] have been developed to run very efficiently on distributed memory computer clusters. More recently, Graphics Processing Units (GPUs), originally developed for rendering detailed real-time visual effects in computer games, have become programmable to the point where they are a viable general purpose programming platform. Dubbed GPGPU for general purpose programming on the GPU, it is currently getting a lot of attention in the scientific community due to the huge computational horsepower of recent GPUs, evident in Figure 6.1 [4–7, 103, 104]. The use of GPGPU techniques as an alternative to distributed memory clusters in MD simulations has become a real possibility.

Until recently, the only way to make use of the GPU's abilities was to carefully cast the algorithm and data structures to be represented as individual pixels being written to an image via *fragment shaders*. In addition to the cumbersome nature of programming this way, there are various other limitations imposed. Perhaps the most severe is that each thread of execution can only write a single output value to a single memory location in a gathered fashion. Scattered writes to multiple memory locations are important in implementing a number of

algorithms, including parts of molecular dynamics. Likely because of this limitation, all the early implementations of MD using GPUs have been based on a mixed approach. The GPU performs those computationally intensive parts of MD that can be implemented in a gather implementation, and the CPU handles the rest [103, 104]. A typical MD simulation implemented on the CPU spends only 50–80% of the total simulation time performing these gather operations. So the speedup by using a mixed CPU/GPU approach is limited to a factor of 2 or 3 at most, even though, as shown in Figure 6.1, the GPU has the ability to perform significantly more floating point operations (FLOPs) per unit time than a CPU, thus leaving room for a dramatic speedup.

In this paper we provide, to our knowledge, the first implementation of a general purpose MD code where all steps of the algorithm are running on the GPU. This implementation is made possible by the use of the NVIDIA® CUDA™ C language programming environment. CUDA provides low level hardware access, avoiding the limitations imposed in fragment shaders. It works on the latest G80 hardware from NVIDIA, and will be supported on future devices [100]. Algorithms developed for this work will be directly applicable to newer, faster GPUs as they become available. After the initial submission of this paper, it came to our attention that J. A. van Meel et al. [7] submitted a similar work nearly simultaneously, where they also used CUDA to put all the steps of MD onto the GPU. The differences in implementation and performance are discussed in subsection 6.3.7.

Early on in the development, it was decided not to take an existing MD code and modify it to run on the GPU. Any existing package would simply pose too many restrictions on the underlying data structures and the order in which operations are performed. Instead, a completely fresh MD code was built from the ground up with every aspect tuned to make the use of the GPU as optimal as possible. Despite these optimizations, every effort was made to develop a *general* architecture in the code so that it can easily be expanded to implement any of the features available in current general purpose MD codes.

This work includes algorithms for a very general class of MD simulations. N particles, in either an NVE or NVT ensemble, are placed in a finite box with periodic boundary conditions,

where distances are computed according to the minimum image convention [15]. Because our own interests are the simulation of non-ionic polymers [24], non-bonded short-range and harmonic bond forces are the only interactions currently implemented in our code. Three major computations are needed in every time step of the simulation: Updating the neighbor list, calculating forces, and integrating forward to the next time step.

Additional interactions such as angular and dihedral terms do not present new conceptual challenges, and can be implemented via adaptations of the algorithms presented here. Electrostatic forces are also required in a wide range of MD simulations. A GPU implementation of the PPPM solver [22] within the current code framework is certainly possible, but left for future work.

6.3 Implementation details

6.3.1 CUDA Overview

Programming for the GPU with CUDA is very different from general purpose programming on the CPU due to the extremely multithreaded nature of the device. For an algorithm to execute efficiently on the GPU, it must be cast into a *data-parallel* form with many thousands of independent *threads of execution* running the same lines of code, but on different data. As a simple example, consider the pseudocode $a_i \leftarrow b_i \cdot c_i + d_i$ which needs to be calculated for all i from 0 to N. In a traditional CPU implementation, this instruction would be inside of a loop that calculates a_0, then a_1 ... one after the other. In contrast, one can imagine that all elements of a_i are calculated *simultaneously* on the GPU as the simplest model of thread execution. Because of this simultaneous execution, the output of one thread cannot depend upon the output of another, which can pose a serious challenge when trying to cast some algorithms into a data-parallel implementation.

In CUDA, these independent threads are organized into *blocks* which can contain anywhere from 32 to 512 threads each, but all blocks must be the same size. Any number of blocks can be run on the device, though optimum performance requires more than a hundred blocks executing in parallel. Each block executes identical lines of code and is given an index starting

from 0 to identify which piece of the data it is to operate on. Within each block, threads are numbered from 0 to identify the location of the thread within the block. Using this indexing scheme, the pseudocode mentioned above could be implemented by using $i \leftarrow \text{bid} \cdot M + \text{tid}$, where bid is the block index, M is the number of threads per block and tid is the index of the thread within the block.

While the simultaneous picture of thread execution on the GPU is illustrative in a conceptual way, implementing efficient algorithms on the GPU requires understanding a little more in detail how the hardware actually executes all N threads. After all, any real hardware can only have a finite number of processing elements that operate in parallel. In particular, a single 8800 GTX GPU contains 16 *multiprocessors*. A single multiprocessor can execute a number of blocks concurrently (up to resource limits) in *warps* of 32 threads. For instance, let's say that there are 3 blocks on a multiprocessor with 128 threads each. In this case, there are 12 warps available for execution. At any given moment, the hardware examines which warps are ready to execute and chooses one. The current instruction of this warp is then executed on the multiprocessor and it moves on to another warp. In this manner, the execution of threads on the device is not so much simultaneous as it is *interleaved*.

There are two important consequences of this execution model. First is that the *smallest* unit of execution on the device is the warp. All 32 threads in the warp *must* execute the same instruction in a data-parallel fashion. Branches (if statements and loops) in the code can lead to different threads executing different instructions. As long as the entire warp follows the same branch, everything is fine. If different threads in the same warp follow different branches, however, the device must serialize this *divergent warp* in an inefficient manner resulting in less than optimal performance. One way that this can be avoided is through the use of *predicated instructions*. If the compiler deems a branch in the code is short enough, it will generate instructions such that all threads in the warp follow *both* branches, but the instructions have a predicate that prevents incorrect output from being generated.

The second consequence is that the device is able to hide memory access latencies. The GPU sits on an expansion board with its own memory (referred to as *global* or *device mem-*

ory) separate from the normal RAM accessible by the CPU. Device memory has a very high bandwidth available, over 70 GB/s, but also has a fairly high delay from the time a memory address is requested to the time it is available. During this latency period, hundreds of arithmetic operations can be performed on a multiprocessor. Thus, the advantage of the interleaved warp execution is that warps which have read their data can be performing computations while other warps running on the same multiprocessor are waiting for their data to come in from global memory, effectively hiding the latency entirely.

On the subject of global memory, there is one additional detail critical to obtaining optimal performance. All memory access within a warp should be arranged such that thread w within the warp accesses the array element base$+w$, where base is a multiple of the warp size. In other words, threads within a warp must access elements along a *row* of a matrix in global memory which has been created with a padding to ensure that the base address is a multiple of the warp size down the columns. When all of these requirements are met, the global memory read is said to be *coalesced*. If the requirements are not met, the application still runs correctly, but memory access performance is reduced by a factor of 10–20. For example, the simple pseucode above has coalesced memory accesses as long as M is a multiple of 32.

Using a coalesced global memory read is the fastest, most efficient way to read device memory, but not all algorithms can be adapted to read memory in such a regular pattern. For random access memory patterns, the hardware provides a *texture unit*, so called because it is used to paint two dimensional images (textures) across three dimensional surfaces when the GPU is used as a graphics output device. Every multiprocessor has its own small texture cache to handle random memory access patterns faster than non-coalesced global memory reads. Textures are just another way of reading global memory and elements can be accessed with one or two dimensional indices. In this work, only one dimensional textures are used and reads are denoted by tex1Dfetch().

The last aspect of CUDA to be aware of for the purposes of this work is that *not all* threads are really executed independently from one another. Only *blocks* are completely independent. For the threads within a single block, there are a few mechanisms by which they can commu-

nicate. First is a barrier mechanism named syncthreads(). Any thread in the block is delayed at this synchronization point until all other threads in the block reach it. Second, each block is given a small area of *shared memory* that exists on the multiprocessor. Any thread in the block can access this shared memory area without the latency associated with global memory reads. There are still a few performance pitfalls even with shared memory though, as it is accessed via 16 banks. For the full details on this and many other aspects of GPU programming not discussed here, interested readers are referred to the CUDA Programming Guide [100].

Most current MD codes developed for execution on a distributed memory cluster [22,52,101] use the Message Passing Interface (MPI) library to perform their computations across many nodes. MPI is similar to NVIDIA's CUDA in some sense: both allow a developer to write one bit of code that is executed many times in parallel on a grid. The similarities end here, however. MPI provides a number of routines for very efficient communication and synchronization between individual processes in the grid. CUDA threads cannot even communicate with each other, and synchronization can only be performed by allowing the kernel call to finish, except for the limited block shared memory and synchronization. Still, many parallel programming concepts commonly used in MPI programming can be applied to a CUDA kernel. Additionally, CUDA and MPI do not need to be considered as two different options for the implementation of a code. A cluster of nodes, each with a GPU, could be constructed and MD code developed that uses CUDA on each GPU with a message passing library to communicate between the nodes. J. E. Stone et al. [6] are working on this using NAMD. In this work, we focus on the core algorithms of MD and optimize them to be as fast as possible on a single GPU.

6.3.2 Short range non-bonded forces

Short range pair forces define the force between any pair of particles as a function of the displacement vector separating them, cutoff to 0 for distances greater than r_{cut}. The forces on all such pairs must be summed to produce the final net force on each of the N particles in the simulation. The standard algorithm [15, 22, 105, 106] is to employ a data structure that lists interacting pairs, the neighbor list, which has already been calculated.

Algorithm 1 Calculate pairwise forces: CPU implementation.

Require: $\vec{F}_k$ is initialized to $\vec{0}$ for all k
Require: NBL_{ji} only stores neighbors where $i < k$
  for all particles i **do**
    $\vec{A} \Leftarrow \vec{R}_i$
    for $j = 0$ to $\text{NN}_i - 1$ **do**
      $k \Leftarrow \text{NBL}_{ji}$
      $\vec{B} \Leftarrow \vec{R}_k$
      $d\vec{r} \leftarrow$ minimum image of $\vec{B} - \vec{A}$
      if $|d\vec{r}| \leq r_{\text{cut}}$ **then**
        $\vec{C} \leftarrow \text{force}(d\vec{r})$
        $\vec{C} + \vec{F}_i \Rightarrow \vec{F}_i$
        $-\vec{C} + \vec{F}_k \Rightarrow \vec{F}_k$
      end if
    end for
  end for

The neighbor list contains particles k that are less than some distance r_{max} from particle i, where $r_{\text{max}} \geq r_{\text{cut}}$. Neighbor list entries are stored in a matrix $\text{NBL}_{ji} \leftarrow k$ with the list of neighbors for each particle i going down a column of the matrix. The number of neighbors in each list varies from particle to particle, so an auxiliary structure NN_i stores the number of neighbors of particle i. Using a matrix to represent the neighbor list deviates from standard practice [15, 22] where a linked list is typically used instead. The matrix is used here because the neighbor list will need to be read in a coalesced way in parallel on the GPU, which is impossible with a linked list.

The calculation of the forces on all N particles is implemented on the CPU following Algorithm 1. Focusing on this CPU implementation for the moment, Newton's third law is used to increment to the forces on particle i and k at the same time, reducing the number of *floating point calculations* that need to be performed by half. However, doing so requires that the all of the scattered memory locations k are accessed via a read-modify-write operation inside inner loop for each particle i. The price of the increased number of memory accesses and their random nature is to drop performance increase from the saved computations down to approximately 1.5 times faster from a maximum of 2 on the CPU.

When designing algorithms on the GPU, the tradeoffs between memory access complexity, number of memory accesses, and number of computations performed are even more significant, since the price of scattered memory access vs. coalesced access is so high. Given this sensitivity, care is taken in writing the pseudocode to call attention to main memory reads (RAM on the CPU, device memory on the GPU), denoted by $\Leftarrow$, and writes, denoted by $\Rightarrow$. Reads and writes from local registers are similarly denoted respectively by $\leftarrow$ and $\rightarrow$.

For clarity of presentation, only a single pair force is calculated for all particles. It is, of course, easy to read in particle type identifiers and use different coefficients in the force calculation depending on what the type pair is.

Algorithm 2 Calculate pairwise forces: GPU implementation

Require: bid is the index of this block
Require: tid is the index of this thread within the block
Require: M is the number of threads in each block
Require: $\lceil N/M \rceil$ blocks are run on the device
Require: NBL_{ji} stores *all* neighbors of each particle

1: $i \leftarrow \mathrm{bid} \cdot M + \mathrm{tid}$
2: $\vec{F}_{\mathrm{sum}} \leftarrow \vec{0}$
3: $\vec{A} \Leftarrow \mathrm{tex1Dfetch}(\vec{R}_i)$
4: $N_{\mathrm{neigh}} \Leftarrow \mathrm{NN}_i$
5: **for** $j = 0$ to $\mathrm{N}_{\mathrm{neigh}} - 1$ **do**
6: $\quad k \Leftarrow \mathrm{NBL}_{ji}$
7: $\quad \vec{B} \Leftarrow \mathrm{tex1Dfetch}(\vec{R}_k)$
8: $\quad d\vec{r} \leftarrow$ minimum image of $\vec{B} - \vec{A}$
9: $\quad \vec{C} \leftarrow \mathrm{force}(d\vec{r})$
10: $\quad$ **if** $|d\vec{r}| > r_{\mathrm{cut}}$ **then**
11: $\quad\quad \vec{C} \leftarrow \vec{0}$
12: $\quad$ **end if**
13: $\quad \vec{F}_{\mathrm{sum}} \leftarrow \vec{C} + \vec{F}_{\mathrm{sum}}$
14: **end for**
15: $\vec{F}_{\mathrm{sum}} \Rightarrow \vec{F}_i$

Casting this calculation into a data-parallel algorithm for the GPU is easily done. Each thread simply calculates the total force on a single particle. Algorithm 2 lists the pseudocode for this implementation. Line 1 defines the indexing scheme for particles, which is the same as used in the simple example in subsection 6.3.1. The choice of this indexing scheme is

important as it sets later lines up for coalesced memory reads, like Line 4, which reads the number of neighbors of particle i. Each thread now needs to loop over all of the neighbors of particle i starting on line 5. Not all particles have the same number of neighbors, so this loop will produce divergent warps. Fortunately, testing shows that the performance loss from this inefficient use of the device is only about 10%.

Moving on to the inner loop, line 6 reads in the index of the neighboring particle from global memory. Since i increases with stride 1 as the thread index increases and i indexes along a row of NBL_{ji} this read is coalesced as long as the proper padding is chosen for the matrix. Line 7 reads the position of the neighboring particle k using the texture unit to take advantage of the texture cache for this random read. The memory access performance of this line is highly dependent on the nature of the layout of the particles in memory. If the neighbors k of particle i are randomly distributed from 1 to N there will be a high cache miss rate and the performance of this memory read can suffer by up to a factor of 4, see Figure 6.3. Such a randomly ordered neighbor list is to be expected in a long running simulation. A solution to this problem is to reorder the particles in memory to improve data locality and the cache hit rate, see subsection 6.3.3.

Continuing, line 9 calculates the force between this pair of particles i, k and adds it to the total in line 13. If this pair of particles is further apart than the cutoff, lines 10 and 11 ensure that the force sum is not modified. With such a simple statement (assigning 0 to a variable) inside the branch, the compiler generates predicated instructions for this operation, verified by reading the assembly output, thus avoiding the performance penalty of divergent warps here. Benchmarks show that this kernel is memory access bound, so any time wasted performing the comparison is hidden within the memory latency.

Options in the implementation of force($d\vec{r}$) are to calculate it directly using floating point operations or to use lookup tables via a texture. With Lennard-Jones potentials directly calculated, performance measurements show that this algorithm's performance is bound by the number of memory accesses it makes. Thus, using a lookup table that introduces more memory reads would slow performance. Potentials with more computationally intensive functional

forms, such as those used in Ref. [6], may perform better when using lookup tables.

Finally, the total force summed for particle i is written to global memory in a coalesced manner at line 15. Notice that the GPU algorithm does *not* take advantage of the Newton's third law optimization used on the CPU. First of all, doing so would require atomic read-modify-write operations on the GPU which are not present for floating point numbers in current hardware. If such features did exist, the increment of $\vec{F}_k$ inside the inner loop still requires a scattered memory access pattern which would slow performance significantly. The additional read and write of $\vec{F}_k$ introduced also leads to an *increase* in the total number of memory accesses performed in an algorithm that is already memory bound. So even if they could be performed at full speed, the performance of the algorithm using Newton's third law would still be slower in the end.

6.3.3 Particle sort

Algorithm 2 reads memory in a random pattern determined by the distribution of the neighbors k of particle i, which are uncorrelated in a long running simulation. If the memory space taken up by the positions of the particles in the system exceeds the cache size, nearly every memory access will then result in a cache miss, requiring the GPU's texture unit to constantly read scattered data from the device memory.

Several sorting techniques have been proposed to solve this issue [105, 107] for MD on the CPU. These solutions rearrange the order in which particle positions are stored in memory so that neighboring particles are also nearby each other in memory. GPART [107] is based on a graph partitioning technique. It arranges particles so that the memory read patterns performed by Algorithm 1 are optimized for the L1 CPU cache. For instance, if the cache line size were such that 4 positions fit in a single cache line, GPART might produce the following neighbor list for a particle: 2 3 4 8 9 10 11 100 101 102 103 ....

Ideal for the serial CPU algorithm, it turns out that GPART generates a poor memory access pattern for the GPU implementation which reads neighbors of M particles simultaneously, a fact that GPART does not take into account. This actually leads to *worse* performance

than randomly ordered data for $N < 20,000$. While it may be possible to develop a modified version of GPART, there is a simpler approach. Early testing of Algorithm 2 was performed on systems of particles in nicely arranged simple cubic arrays. Performance tests showed memory transfers approaching device limits, and a successful sort algorithm was developed that mimics this pattern for arbitrary systems. An even better option is to reorder the particles based on the path of a space filling curve [108, 109], which have received a lot of attention lately for efficient database searches on multi-dimensional data sets. The Hilbert curve is chosen for this work because it has the best locality preserving properties [108]. Ref. [107] mentions the possibility of using a Hilbert curve for the data sort algorithm, but ignores it as a viable option in favor of another algorithm, RCB.

Ref. [109] discusses, in detail, a recursive algorithm for generating the Hilbert curve on the fly and assigning an index to each point in a point cloud. This index is then used as the primary key in a database record. The algorithm avoids ever computing the entire Hilbert curve so that large data sets which do not fit into system RAM can be indexed. In this work, all particles do fit in memory, so a simpler approach can be adopted, which we call the space-filling curve pack (SFCPACK) algorithm. SFCPACK consists of the following steps.

1. Divide the simulation box into cells $\approx 1\sigma$ wide.
2. Place each particle into its corresponding cell.
3. Generate a traversal over the cells using the recursive algorithm in Ref. [109]
4. Loop over the cells in this traversal order, making an ordered list of particles visited
5. Reorder the particles based on the list

6.3.4 Neighbor list generation

The objective of the neighbor list generation algorithm is to examine the current positions of all N particles and build a list for each containing those particles separated by a minimum image distance [15] less than the cutoff. This can be trivially done by an all vs. all comparison,

but doing so requires $O(N^2)$ execution time making simulations of large systems prohibitively time consuming. Algorithmic improvements on the CPU have reduced the time to scale as $O(N)$ in the average case [22, 105, 106]. Implementing this more complicated neighbor list algorithm on the GPU poses the biggest challenge in developing a general purpose MD code running fully on the device. It requires the building of variable length lists using scattered memory writes, which cannot be realized with the fragment shader approach, but can be easily performed using CUDA.

To improve the overall performance of the entire system, MD codes such as LAMMPS [22] employ a standard trick. The cutoff distance for the neighbor list is chosen as $r_{\max}$, greater than the value of r_{cut} used for the pair forces. Then, the neighbor list only needs to be updated when any particle has moved a distance more than $\frac{1}{2}(r_{\max} - r_{\text{cut}})$, which is usually every 10 or more time steps.

Actually updating the list in $O(N)$ time involves looping through the N particles and placing them into N_{cell} cells of width r_{cell}, called *binning* the particles. When building the neighbor list, only the particles in the set of neighboring cells need to be considered [105, 106]. When r_{cell} is chosen to be equal to $r_{\max}$, 27 bins need to be read for each particle, but a significant fraction of these particles will not make it into the neighbor list. Floating point computations can be saved at the cost of increased memory accesses (125 neighboring cells need to be accessed, many of which are empty) by choosing $r_{\text{cell}} = \frac{1}{2}r_{\max}$ [105, 106]. Performance tests were made on the GPU, showing that the additional memory accesses cost more than the extra floating point operations, so r_{cell} will be fixed at $r_{\max}$ for this work.

After the particles have been placed in their cells, updating the neighbor list matrix is accomplished on the CPU with the pseudocode in Algorithm 3. It is included so that comparisons can be drawn with the GPU implementation.

The first portion of the algorithm, binning the particles, is challenging to implement efficiently on the GPU. Our implementation copies the particle positions from the device, bins them on the CPU and then copies the resulting cell lists back to the device. The binning process is slow and the overhead from the memory copies to and from the device is appreciable.

Algorithm 3 Neighbor list build: CPU implementation

for all particles i **do**
  $\vec{A} \Leftarrow \vec{R}_i$
  $A_{\text{cell}} \leftarrow$ cell containing $\vec{A}$
  $N_{\text{neigh}} \leftarrow 0$
  for all 27 cells C in the neighborhood of A_{cell} **do**
    for all particles j in cell C **do**
      $\vec{B} \Leftarrow \vec{R}_j$
      $d\vec{r} \leftarrow$ minimum image of $\vec{B} - \vec{A}$
      if $|d\vec{r}| \leq r_{\max}$ **and** $i \neq j$ **then**
        $j \Rightarrow \text{NBL}_{N_{\text{neigh}},i}$
        $N_{\text{neigh}} \leftarrow N_{\text{neigh}} + 1$
      end if
    end for
  end for
end for
$N_{\text{neigh}} \Rightarrow \text{NN}_i$

But, the whole operation only needs to be performed once approximately every 10 time steps, reducing the total overhead in a full simulation to 6%.

There are two methods that can be used to move this step to the GPU. First, the newer generation G92 graphics cards add the ability to perform read-modify-write operations atomically, which could be used to implement the binning on the GPU. Lacking one of these GPUs, we could not try this method. Another method involving double buffered partial updates of the cell list is presented in Ref. [7]. There was not sufficient development time available to try this method here before publishing. Either of the two could significantly reduce the 6% overhead mentioned previously, a reasonable but not significantly large speedup.

The implementation of the second step on the GPU is listed in Algorithm 4. Each block executed on the device handles the calculation of all of the neighbor lists for particles i of a single cell. The advantage of this layout is that all particles in a given cell must be compared against the same set of particles in neighboring cells. Thus all of the memory operations reading particle positions need only be done on a per block basis and then accessed via shared memory. Line 1 of the algorithm starts this off by reading in the particle index i from the

$\text{CELL}_{a,C}$ array in main memory. Like the neighbor list matrix NBL_{ji}, $\text{CELL}_{a,C}$ is a matrix listing the particles j belonging to bin C. The difference is that the particle indices are listed along the *rows* of the matrix so that the reads on lines 1 and 6 will be coalesced (as long as the matrix has the correct padding).

Algorithm 4 Neighbor list build: GPU implementation

Require: bid is the index of this block
Require: tid is the index of this thread within the block
Require: M is the number of threads in each block
Require: $M \geq$ the largest number of particles in any given cell
Require: N_{cell} blocks are run on the device
Require: $\vec{B}_j$ and K_j are stored in shared memory

```
1:  i ⇐ CELL_{tid,bid}
2:  N_neigh ← 0
3:  A⃗ ⇐ tex1Dfetch(R⃗_i)
4:  for all 27 cells C in the neighborhood of bid do
5:      syncthreads()
6:      K_tid ⇐ CELL_{tid,C}
7:      B⃗_tid ⇐ tex1Dfetch(R⃗_{K_tid})
8:      syncthreads()
9:      if not empty(i) then
10:         for j = 0 to M − 1 do
11:             if empty(K_j) then
12:                 break
13:             end if
14:             C⃗ ← B⃗_j − A⃗
15:             dr⃗ ← minimum image of B⃗ − A⃗
16:             if |dr⃗| < r_max and K_j ≠ i then
17:                 K_j ⇒ NBL_{N_neigh,i}
18:                 N_neigh ← N_neigh + 1
19:             end if
20:         end for
21:     end if
22: end for
23: N_neigh ⇒ NN_i
```

Line 3 reads in the position of particle i for which this thread is going to build the neighbor list. As the loop on line 4 goes over all of the neighboring cells, the particle indices K_j and positions $\vec{B}_j$ are loaded into shared memory on lines 6 and 7. Particle positions are read using

the texture unit to take advantage of the cache. The synchthreads() barriers surrounding these memory reads are needed to ensure that there are no race conditions when accessing shared memory. Line 9 skips computations if the particle i is actually an empty entry in the cell. This statement will produce divergent warps, but it prevents a large number of FLOPs from being wasted on empty cell elements. It cannot be placed any earlier in the code, since all threads in the block need to participate in the data loading done on lines 6 and 7.

Continuing on, line 10 starts a loop over all particles in cell C. This loop is done in every thread because every particle i must be checked against every other particle in the neighboring cells for inclusion into the neighbor list. Line 11 immediately checks if the current particle index K_j is an empty value, quitting the loop on line 10 if it is. Performing this early-out check inside the loop is important for performance, since a typical cell may have 30 particles in it, but the maximum has 60. Finally, lines 14 through 18 check the current pair of particles for inclusion into the neighbor list. All threads in any given warp in the block are reading the same indices of K_j and $\vec{B}_j$ simultaneously, so the shared memory broadcast will be invoked and there are no bank conflicts to slow shared memory read performance [100]. Scattered writing to the neighbor list on line 17 is not coalesced and thus will not perform optimally on the device. Fortunately, the bottleneck in this algorithm is the floating point operations so the inefficient memory write does not change its overall performance significantly.

6.3.5 Integration

The explicit Nosé-Hoover integration method from Ref. [110] is implemented on the GPU to move particle positions and velocities forward one time step in the NVT ensemble. Implementation of this method is straightforward and not provided here. It is important, however, that this step is done *on the GPU*, even if the computation takes very little time on the CPU. When done on the CPU, forces calculated on the GPU must be copied to the CPU, the integration performed, and then the new particle data copied to the GPU. All of these memory copies amount to a significant fraction of the simulation time, as they are done on every time step. Additionally, the speedup factor of 20 executing on the GPU prevents the execution time

of the integration on the CPU from becoming a bottleneck.

The NVE ensemble is similarly implemented using a velocity Verlet method [22].

Multiple time step methods [111] can be used to reduce computational cost in some types of MD simulations containing both slowly and quickly varying forces. Such methods can be implemented on the GPU in a straightforward manner, identical to any CPU implementation. None are implemented or benchmarked for this work. However, any relative reduction in computational cost on the CPU by using one of these methods would translate in a similar reduction on the GPU.

6.3.6 Bond forces

Harmonic bond forces are implemented on the GPU for the same reasons as the integration. Algorithm 2 is modified to read a different NBL_{ji} that includes a list of all particles k *bonded* to i instead of the neighbor list. Speedups of a factor of 75 compared to the CPU implementation are achieved.

Angle and dihedral terms are not implemented in this work, but could be achieved in a similar fashion requiring only a slightly more complicated list structure. Similar speedups are expected.

6.3.7 Brief comparison to other recent works

Refs. [6, 7] also implement Molecular Dynamics using CUDA on the same hardware as is used here. Both work around the complexity of generating a neighbor list by storing particle positions in a cell data structure and using that directly in the pair force computation on the GPU.

J. E. Stone et al. [6] target large biomolecule simulations with their implementation. It only computes pair forces on the GPU, leaving all other tasks for the CPU. The pair potential they use is much more computationally intensive than the simple Lennard-Jones potential we use here. So instead of performing the computations directly on the GPU, they use a texture as a lookup table to interpolate pre-calculated terms. No performance comparisons can be made

with this work because the potentials and systems benchmarked vary significantly. Another notable difference in their work is that they have linked their CUDA implementation of MD directly into NAMD and can thus distribute jobs across a cluster of nodes with GPUs.

The implementation by J. A. van Meel et al. [7] is closer to our own, modeling simple Lennard-Jones particles and fully implementing every step of MD on the GPU. A direct performance comparison can be made to quantify the tradeoffs between the neighbor list and cell based approaches. At parameters relevant to our work, $\rho = 0.4\sigma^{-3}$ and $N = 100,000$ they report 0.02 seconds of computation time per step, or 50 time steps per second. Configuring our software to implement the Lennard-Jones liquid with identical parameters, we measure 160 time steps per second, 3.2 times faster.

6.4 Performance Measurements

6.4.1 Hardware

All single CPU / GPU benchmarks are run on a Dell Precision Workstation 470 with a 3.0 GHz 80546K Xeon processor and 1 GB of RAM. The original graphics card in this system was upgraded to a NVIDIA GeForce 8800GTX manufactured by EVGA® with the standard core clock speed of 575 MHz. The system dual boots Microsoft® Windows® XP and Gentoo Linux running the latest 2.6.21 AMD64 kernel. CUDA 1.1 is installed in both operating systems. Under Windows, code is compiled using Visual Studio Express 2005 with the compiler optimization options "/arch:SSE2 /Ox /Ob2 /Oi /Ot /Oy /fp:fast". Linux executables are produced with gcc version 4.1.2 using compiler optimization options "-march=nocona -O3 -funroll-loops".

CPU benchmarks are approximately 20% faster in Linux compared to Windows. GPU benchmarks are approximately 5% faster in Windows compared to Linux. The fairest comparison possible is given in this work by presenting CPU benchmark results obtained in Linux and GPU benchmark results obtained in Windows.

Lightning, a cluster on the ISU campus, is used for the LAMMPS cluster benchmarks. It has 90 compute nodes, each of which contain two dual-core AMD Opteron™ 280 processors

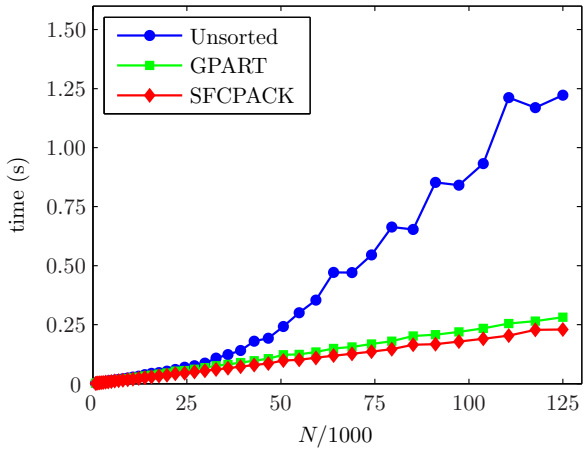

Figure 6.2 Time to calculate Lennard-Jones forces for all N particles on the CPU plotted vs. system size. Results are shown with various sorting algorithms applied to the particles.

and 8 GB of RAM. Nodes are interconnected with a high speed, low latency InfiniBand® network. PathScale™ 2.4 is used to compile LAMMPS on Lightning. In order to get the best performance possible out of LAMMPS for the fairest comparison, the new optimized Lennard-Jones routines from style_opt are used.

6.4.2 Lennard-Jones force calculation

To test the performance of Algorithm 1 and Algorithm 2 in computing Lennard-Jones forces, particle systems are created with N particles randomly placed in a box with side lengths chosen so that the number density of particles is $n = 0.382\sigma^{-3}$. The force cutoff is set to $r_{\text{cut}} = 3.0\sigma$, which is typical in coarse-grained simulations [24] and r_{max} is set to a reasonable value of 4.0σ. Relative GPU/CPU performance differs little for various values of r_{max}, so the precise value is irrelevant. The random placement of the particles is to emulate a typical time step in a simulation of a Lennard-Jones liquid. With the chosen cutoff values, there are 102 neighbors for each particle on average. While the particles are randomly placed, the same random seed is used each time so that the GPU and CPU benchmarks at the same N are calculating forces on exactly the same system of particles. Timing is performed by generating the neighbor list

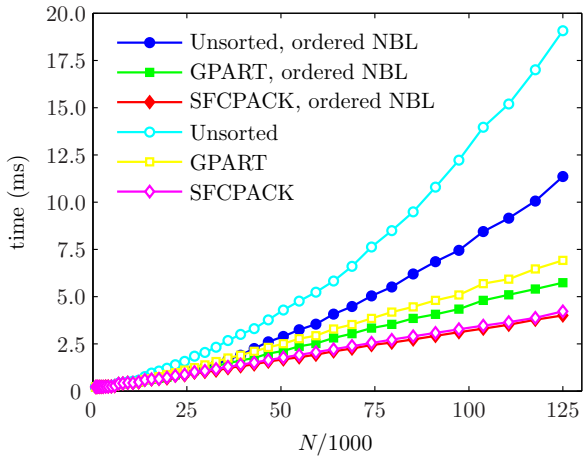

Figure 6.3 Time to calculate Lennard-Jones forces for all N particles on the GPU plotted vs. system size. Results are shown with various sorting algorithms applied to the particles. Open symbols indicate times were obtained with unordered neighbor lists, and closed symbols indicate the neighbor list was ordered.

once and then running the force calculation repeatedly until five seconds have elapsed or the calculation has been done five times, whichever comes first.

Benchmark runs are performed on the CPU with the particle data unsorted, sorted by GPART [107], and sorted by SFCPACK. Results are plotted in Figure 6.2. Performance is improved greatly with the sorting algorithms applied, especially for large N due to the more effective use of the memory cache. It is also worth noting that the much simpler SFCPACK algorithm beats the performance of GPART [107].

On the GPU, the performance of Algorithm 2 is sensitive to the choice of the block size M. Even though the same number of calculations and memory accesses are performed regardless, memory access patterns, register read after write dependencies, and the warp occupancy of the kernel running on the device can all change the running time [100]. It is impossible to predict which block size will lead to the best performance, so benchmarks on the GPU are performed at *all possible* block sizes from 32 to 448 in steps of 32, and the fastest time is chosen for the benchmark result. It would make no sense to select a value for M that is not a multiple of 32 since that is the smallest unit of execution on the device. The fastest block size for each system

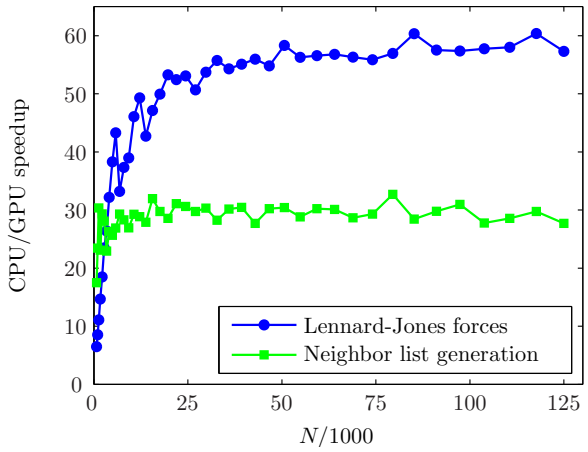

Figure 6.4 CPU time / GPU time for the Lennard-Jones force calculation and neighbor list generation plotted vs. system size. Particles are sorted using the SFCPACK algorithm in these benchmarks.

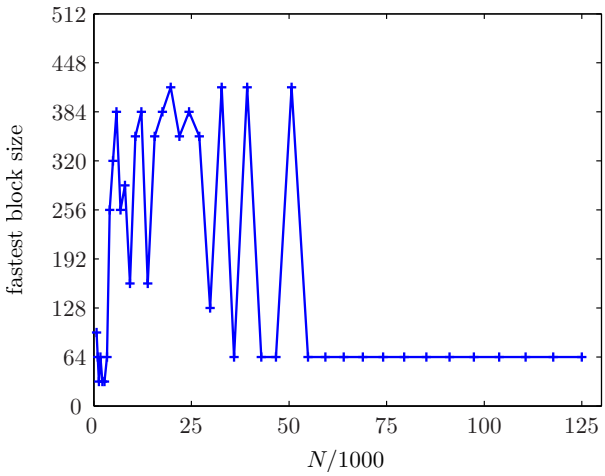

Figure 6.5 Best performing block size M for the GPU implementation of the Lennard-Jones force calculation plotted vs. system size.

size is recorded in Figure 6.5. When performing actual MD simulations of a specific system, these results can be used to determine which block size to use for the best performance.

Like the results obtained on the CPU, performance of the Lennard-Jones force calculation on the GPU is sensitive to the order of the particles in memory, so benchmarks are performed with particles sorted by the various algorithms as before. Additionally, and unlike the CPU, performance is also sensitive to the order of particles *in the neighbor list*. When the neighbor list is generated by the simple $O(N^2)$ method, $\mathrm{NBL}_{ji} < \mathrm{NBL}_{j+1,i}\ \forall j$. Even though the neighbors k may be randomly distributed from 0 to $N-1$, this forced ordering in the neighbor list improves the locality of memory reads over a randomly ordered neighbor list. On the contrary, the more efficient $O(N)$ neighbor list does not generate ordered neighbor lists. Instead, it generates 27 independent sorted sequences in NBL_{ji} as j goes from 0 to $\mathrm{NN}_i - 1$. While it is possible to use merge sort techniques to generate a fully ordered neighbor list here, the performance hit is significant. In order to evaluate the tradeoffs between an ordered neighbor list vs. an unordered one, the force computation is benchmarked using each on the GPU. Timing results for all these cases are provided in Figure 6.3. The same trend seen on the CPU (SFCPACK is faster than GPART which is faster than unsorted) is repeated here. As expected, force computations done using ordered neighbor lists are faster than with unordered ones. Although, with the SFCPACK sort the difference is only a few percent. With these results in hand, the clear choice for implementing the fastest MD on the GPU is to use the fast, unordered neighbor list generator and the SFCPACK sort.

The fastest evaluations of the pair force calculations for the CPU and GPU, obtained with the SFCPACK sorted particles, are compared in Figure 6.4. At 60 times faster for large enough systems, the GPU is an incredibly powerful tool for performing the pair force computation. While crunching numbers in the linear performance region ($N > 15000$), the device is reading and writing memory at a rate of 60–70 GB/s which is close to theoretical peak device capability [100]. At the same time, only about 100 GFLOP/s are performed which is far from its theoretical peak of 340 GFLOP/s indicating that many of the multiprocessors sit idle with all of the active warps waiting on the global memory access latency. Thus, short

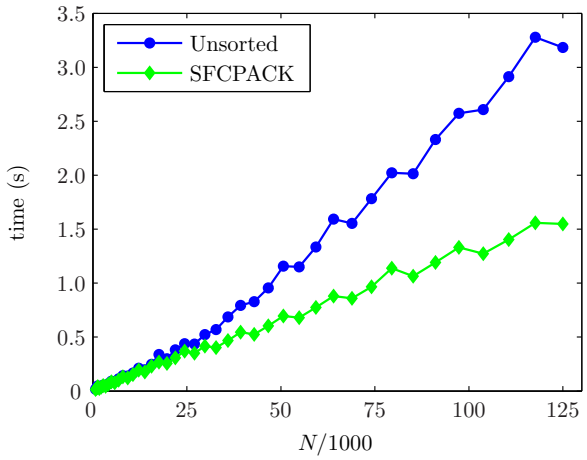

Figure 6.6 Time to generate the neighbor list for all N particles using Algorithm 3 on the CPU plotted vs. system size. Results are shown with and without the SFCPACK sort applied to the particle data.

range pair forces that require more floating point operations than Lennard-Jones could also be implemented in Algorithm 2 without a significant loss in performance compared to the values presented here.

6.4.3 Neighbor list generation

The neighbor list generation code in Algorithm 3 and Algorithm 4 is benchmarked on the same random particle systems used to benchmark the pair forces. The parameter $r_{\max}$ is not changed from 4.0σ. Removing the checks that only update the neighbor list when a particle has moved a sufficient distance, the same neighbor list is generated over and over again until five seconds has elapsed or the neighbor list has been generated five times. As the neighbor list generation algorithms read particle positions randomly from the data arrays based on the cell list, they also benefit from the particle data sort. Timings for these benchmarks are plotted in Figure 6.6 and Figure 6.7, which show the total time taken to bin the particles and update the neighbor list matrix. Execution time clearly increases linearly for both the CPU and GPU implementations with the total speedup on the GPU hovering around 30, as shown

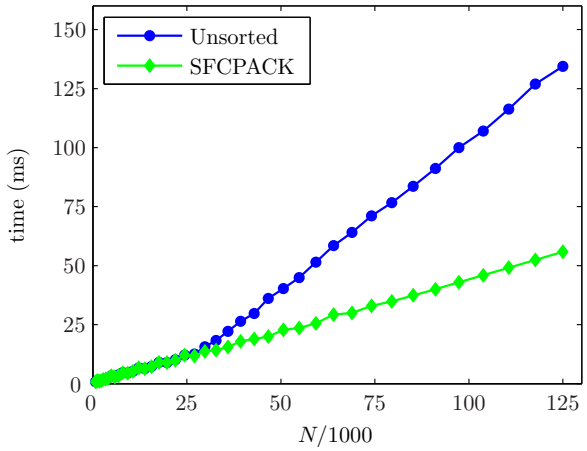

Figure 6.7 Time to generate the neighbor list for all N particles using Algorithm 4 on the GPU plotted vs. system size. Results are shown with and without the SFCPACK sort applied to the particle data.

in Figure 6.4. While not quite as impressive a speedup as with the Lennard-Jones forces, the neighbor list generation is still significantly faster on the GPU.

6.4.4 Benchmark of MD simulations

For a practical comparison with an existing software solution, benchmarks of the full MD simulation are run using both the GPU implementation on a single GPU and LAMMPS [22] running on the Lightning computer cluster. It should be expected that some differences might be obtained by testing other general purpose MD packages, but in this paper all comparisons are done with LAMMPS as this is the code we use in our applications.

The neighbor list generation and pair force computations algorithms interact with $r_{\max}$ in different ways. If the value is chosen too small, the neighbor list will need to be updated every other time step, slowing performance significantly. A larger value leads to fewer neighbor list updates, but then the length of the neighbor list gets larger and more forces need to be computed slowing down overall performance again. In all benchmarks presented here, short test runs with $r_{\max}$ varying in steps of 0.1 are performed to determine the optimal value.

6.4.4.1 Lennard-Jones liquid

A full simulation of a Lennard-Jones liquid is implemented with the GPU algorithms presented in this work. Initial conditions are prepared by starting from a random arrangement of N particles at density $n = 0.382\sigma^{-3}$, then running the simulation in the NVT ensemble at $\frac{k_B T}{\varepsilon} = 1.2$ for 50,000 time steps to equilibrate the system. An additional 50,000 time steps are then timed for the benchmark. The timed runs for both the GPU and LAMMPS both start from the same prepared initial condition. For all simulations, $r_{\text{cut}} = 3.0\sigma$.

On the GPU, the optimal value of r_{max} is found to be 3.8σ for this system. Additionally, the block size for the force computation is chosen from Figure 6.5 and SFCPACK is used to resort the particles every 1,000 time steps. LAMMPS performed best with $r_{\text{max}} = 3.4\sigma$ and was timed running on various numbers of processor cores up to 40.

Results of these tests are shown in Figure 6.8 and summarized in Table 6.1. As expected from the previous comparisons, Figure 6.4, the GPU performs at the same speed as around 36 processor cores on Lightning.

6.4.4.2 Coarse-grained polymer systems

To show the versatility of our GPU implementation, another test case is implemented that uses a portion of the polymer model from Ref. [24]. In short, polymers are built with A and B particles connected by harmonic bonds to form a chain: $A_{10}B_7A_{10}$. Non-bonded interactions are implemented with forces derived from Lennard-Jones potentials:

$$U_{AA}(r) = 4\varepsilon \left(\frac{\sigma}{r}\right)^{12}$$

$$U_{BA,BB}(r) = 4\varepsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right]$$

Preparation and timing of these systems is performed as before with the Lennard-Jones liquid, with all parameters identical.

The benchmark timings for various system sizes are plotted in Figure 6.9 and summarized in Table 6.1. On the GPU, absolute performance numbers are slightly lower when compared to the Lennard-Jones liquid system from the resulting overheads of the bond force sum. LAMMPS

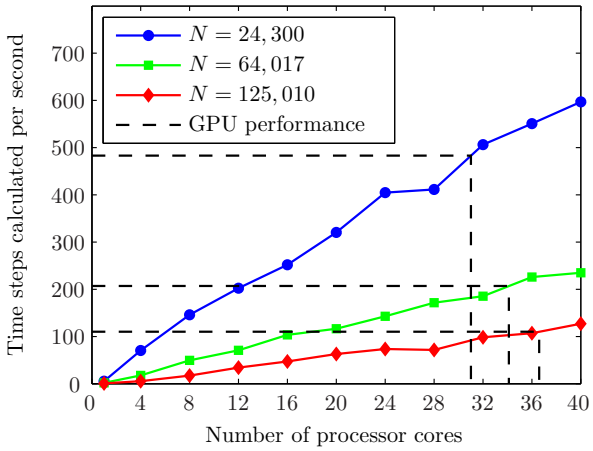

Figure 6.8 Benchmark of LAMMPS simulating the Lennard-Jones liquid model for various system sizes on Lightning. The dotted lines mark the performance of the GPU running the same simulation.

System	Lennard-Jones liquid	Polymer model
Density	$0.382\sigma^{-3}$	$0.382\sigma^{-3}$
Average neighbors per particle	43.2	43.2
CPU time / particle / time step (LAMMPS)	2.64 μs	2.34 μs
GPU time / particle / time step	0.0778 μs	0.0806 μs

Table 6.1 Average performance results summary.

on the other hand is actually slightly *faster* at executing the polymer system. Examination shows that the Lennard-Jones liquid requires significantly more time spent on inter-node communication. These two effects combine to bring the overall comparative performance of the GPU down a bit to 32 processor cores on Lightning.

6.5 Numerical precision tests

Current GPUs only offer support for single precision floating point arithmetic, and not all operations meet the IEEE 754 standards [100]. It may be argued that the precision of results obtained via MD simulation on the GPU are thus suspect. To demonstrate that this is not the case, simulations of the polymer system are performed on the CPU and GPU using single

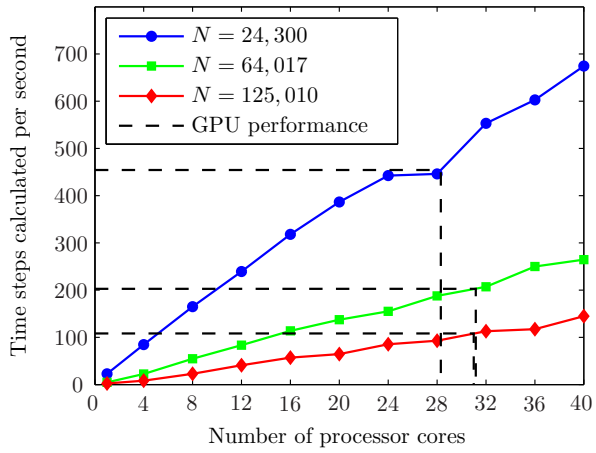

Figure 6.9 Benchmark of LAMMPS simulating the polymer model for various system sizes on Lightning. The dotted lines mark the performance of the GPU running the same simulation.

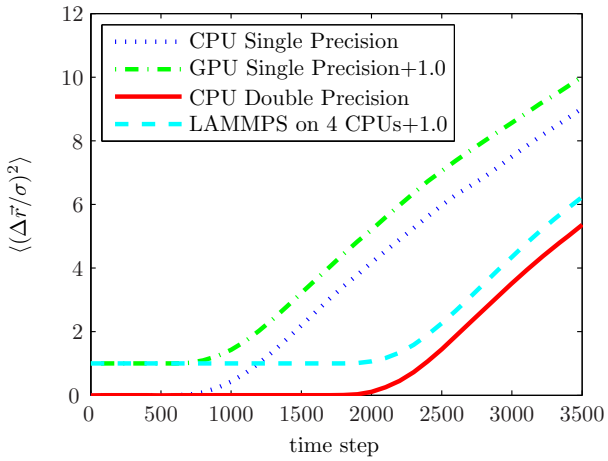

Figure 6.10 Deviation of simulation trajectories from a baseline generated by LAMMPS running on a single processor. Single and double precision modes on the CPU and the single precision GPU calculations are compared to the baseline. Data from a LAMMPS run on four processors is also presented. Two of the four curves have their 0's artificially shifted to visually separate them from their counterparts.

precision math, on the CPU using double precision math, and using LAMMPS on Lightning. Identical initial conditions are used in all runs. The equilibrated initial condition is produced via the same method used in subsection 6.4.4 with $N = 24,300$, and then all test cases continue the simulation from that point. LAMMPS and the code developed for this work use different implementations of the NVT integrator and thus cannot be compared. Continuation runs are instead performed in the NVE ensemble, where both codes use a Velocity Verlet integrator [22].

Particle positions are dumped every 100 time steps in each simulation run performed. The $\langle(\Delta\vec{r}/\sigma)^2\rangle$ deviation is computed between the particle positions of two different simulations at identical time steps. Using a LAMMPS double precision run on a single processor as a baseline for comparison, Figure 6.10 shows the results. Initially, LAMMPS and the code developed in this work do indeed calculate the same trajectory. Using double precision computations on the CPU, 2000 time steps are performed before the trajectories start to differ significantly. Single precision calculations on the CPU are slightly less precise with the trajectories starting to diverge at 1000 time steps. The GPU single precision implementation is no worse than the CPU, diverging at exactly the same point.

To demonstrate what one *should expect* of a precise MD trajectory, the same simulation run is performed using LAMMPS again, but this time on 4 processor cores in parallel. In this case, the trajectory starts to deviate at 2000 time steps, which is no different from the double precision calculation on the CPU. These two simulations starting from identical initial conditions can vary so drastically even when executed with the same code because forces on the particles are summed in a different order on 1 CPU vs. 4 CPUs. With floating point arithmetic, summing identical values in a different order can produce different results. The slightest difference in a computed force introduces a difference in the particles' positions between the two simulations at the next time step. These differences then compound time step after time step until the trajectories vary significantly. A typical MD simulation will integrate millions of time steps, so a deviation at 1000 vs. 2000 steps is irrelevant.

Energy conservation during an NVE simulation run is a more rigorous metric to measure the precision of MD simulations. We check that energy is conserved on the GPU, but do

not provide a detailed quantitative study here. Future GPUs will include support for double precision [100], so the capability will eventually be there for those simulations that need the extra precision.

6.6 Conclusions

We have presented a general purpose MD simulation fully implemented on a single GPU. We have compared our GPU implementation against LAMMPS running on a fast parallel cluster, see Figure 6.8, and we have shown that the GPU performs at the same level as up to 36 processor cores. Smaller, less power hungry, easier to maintain, and inexpensive compared to such a cluster, GPUs offer a compelling alternative. And this is only the beginning. As Figure 6.1 shows, the trend towards faster GPUs is inexorable, allowing a simple component upgrade to enable even more demanding MD simulations. The fact that all future NVIDIA GPUs will support the CUDA environment [100] warrants the generality of the implementation presented here.

Current technology allows even faster MD simulations in a single workstation. Up to four GPUs can be hosted in a single workstation potentially bringing the the power of a 144 core cluster to the desktop at a fraction of the cost. While GPUs do not currently provide a viable alternative to supercomputers able to simulate systems with more than a million particles, we expect that GPUs will very soon become a commonplace tool for smaller simulations. The only major element missing for many types of simulations is the implementation of electrostatic forces, which we leave for future work.

6.7 Acknowledgments

This work is funded by NSF through grant DMR-0426597 and by DOE through the Ames lab under contract no DE-AC02-07CH11358.

CHAPTER 7. DESIGN OF POLYMER NANOCOMPOSITES IN SOLUTION BY POLYMER FUNCTIONALIZATION

A paper currently in progress that will be submitted for publication soon

Joshua A. Anderson and Alex Travesset

7.1 Abstract

Polymer nanocomposites, materials combining polymers and inorganic components such as nanoparticles and nano-sized crystallites, have attracted significant attention in recent years. A successful strategy for designing polymer nanocomposites is polymer functionalization via attaching functional groups with specific affinity for the inorganic component. In this paper, a systematic investigation of polymer functionalization for nanocomposite design in solution combining molecular dynamics simulations and theoretical arguments is presented. Optimal polymer architectures and concentrations are identified appropriate for different applications, along with in-depth analysis on the origin and stability of the resulting phases.

7.2 Introduction

Polymer nanocomposites, materials combining polymers and inorganic components such as nanoparticles and nano-sized crystallites, have been identified as suitable candidates for novel photovoltaic and hydrogen fuel cells, membranes for gas (e.g., CO_2) separation, self-healing coatings, novel materials for drug delivery, sensing and in vivo imaging, fabrication of artificial bone matrix as well as smart materials with novel mechanical, rheological, optical and other properties [2]. One of the most fundamental challenges in polymer nanocomposite design is

to achieve control of the orientation and location of both the inorganic components and the polymer so that the resulting material will exhibit the desired properties.

Block copolymer melts with dispersed nanoparticles provide the most investigated examples of polymer nanocomposites both experimentally [112–121] and theoretically [122–128]. Despite some remarkable successes, some major issues remain open as discussed, for example, by Ganesan [129]. One of them is the long equilibration times involved, which often results in the system being trapped in a long lived metastable state. A possibility to overcome this problem is to disperse the soluble inorganic components into a thermoreversible polymeric gel (in solution), with examples such as calcium phosphate [99,130] and silica nanoparticles [98] in a Pluronic [33] gel. In this latter case, the thermoreversibility of the gel allows thermodynamic equilibrium to be reached as the nanoparticles and polymer are mixed at low temperatures where the solution has low viscosity and short equilibration times.

In this paper, soluble inorganic components dispersed into a gel-forming block-copolymer consisting of hydrophilic (soluble) and hydrophobic (insoluble) blocks are studied. In a simple scenario, the inorganic components added into the solution do not have any affinity for the hydrophilic blocks, but attract each other (due to van der Waals or chemical interactions). In a previous paper [12], it is established that such a system will not self-assemble into a composite. Either the inorganic components freely diffuse within the solvent or, if their mutual attraction becomes strong enough, phase separate from the polymer matrix. There is no intermediate regime where the polymeric matrix is able to coerce the aggregation of the inorganic phase sufficiently to impose order on it. Therefore, some affinity between the inorganic component and the hydrophilic blocks of the polymer is necessary in order to successfully assemble a composite from solution. A recent study using different interaction potentials has reached the same conclusions, providing more evidence for this result [131].

There has been a remarkable progress on nanoparticle design in recent years [132]. In many cases, the *nanoparticles* themselves are functionalized with short polymers in order to solubilize and/or prevent agglomeration. In previous papers [12, 13], a strategy is proposed where the *polymer* is functionalized with the addition of end blocks having a high affinity for

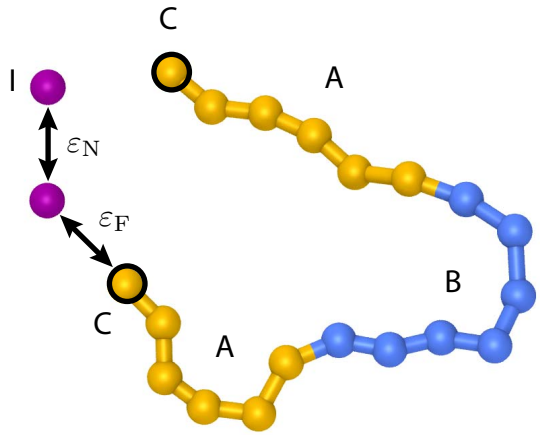

Figure 7.1 Polymer model

the inorganic component. While there is a considerable repertoire of functional groups that can be attached to polymers, Pluronics [8] in particular, polymer functionalization has not received as much attention as nanoparticle functionalization. Yet based on previous results [12, 13] as well as the ones presented in this paper, an approach where the polymers are functionalized for specific nanoparticles or other inorganic components is a very promising pathway to enable the design of polymer nanocomposite materials.

The goal of this paper is to present a systematic study of polymer functionalization for nanocomposite design, combining molecular dynamics (MD) and theoretical arguments to provide a thorough understanding of the opportunities and limitations that polymer functionalization brings. Particularly, the focus is on the effects of polymer architecture and concentration, while also exploring the full region of parameter space corresponding to all inorganic component affinities. At high affinities the inorganic component results in a unique form of crystalline order not found before.

7.3 Simulation Details

7.3.1 Model

Polymers and inorganic particles are modeled with coarse-grained beads in an implicit solvent. Individual polymers are $CA_nB_mA_nC$ symmetric block polymers, where the A and C beads are hydrophilic and B beads are hydrophobic. Harmonic bonds connect the monomers of the polymer into a linear chain.

Additionally, Lennard-Jones like pair potentials are applied between all non-bonded pairs of beads. Hydrophobic interactions are attractive, with a potential energy given by

$$U_{\mathrm{BB}} = 4\varepsilon_{\mathrm{P}} \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] , \tag{7.1}$$

while hydrophilic interactions are purely repulsive with

$$U_{\mathrm{AA,AB,CA,CB}} = 4\varepsilon_{\mathrm{P}} \left(\frac{\sigma}{r}\right)^{12}, \tag{7.2}$$

where r is the distance between the two beads in the pair.

The inorganic particles I are modeled by single beads. Alone, the inorganic particles are assumed to have a tendency to aggregate with a characteristic energy scale ε_{N}, so they are modeled with an attractive potential

$$U_{\mathrm{II}} = 4\varepsilon_{\mathrm{N}} \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] , \tag{7.3}$$

and they have an affinity to the functional end blocks of the polymer C with a different energy scale ε_{F}

$$U_{\mathrm{IC}} = 4\varepsilon_{\mathrm{F}} \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] . \tag{7.4}$$

Other interactions between the polymer and inorganic beads are purely repulsive

$$U_{\mathrm{IA,IB}} = 4\varepsilon_{\mathrm{P}} \left(\frac{\sigma}{r}\right)^{12}. \tag{7.5}$$

For more details and justification of this model, see our previous publications [12,24]. A visual summary of the parameters is given in Figure 7.1.

All of these pair potentials are cut off at $r_{\mathrm{cut}} = 3.0\sigma$. The value of ε_{P} is fixed at $\frac{1}{1.2}\varepsilon$ while ε_{F} and ε_{N} are free parameters over which phase diagrams are computed. The remaining unknown quantities, ε and σ, define the two fundamental units of energy and length in the simulation with the mass of the particles making the third.

N_{poly} polymers and $N_{\mathrm{inorganic}}$ I beads are initially placed randomly in a cubic simulation box with periodic boundary conditions. Molecular dynamics simulations are carried out from this initial configuration in the NVT ensemble via Nosé-Hoover dynamics [17]. Some runs are performed at a constant kinetic temperature $k_{\mathrm{B}}T/\varepsilon = 1.0$, but most are started higher and slowly cooled to this point to improve equilibration.

Early simulations in this work are performed using the LAMMPS [22] software package, while HOOMD [3] is used for the most recent.

7.3.2 Observables

The concentration of polymers in solution is calculated from the polymer packing fraction ϕ_P defined by

$$\phi_P = \frac{\pi N_{\mathrm{poly}} N_{\mathrm{mon}}}{6(L/\sigma)^3} \ , \tag{7.6}$$

where N_{mon} is the number of monomers in each polymer chain and L is the length of each side of the cubic simulation box. Note that ϕ_P is defined only as a function of the polymers in the simulation box and does not depend on the number of inorganic particles $N_{\mathrm{inorganic}}$. This is done so that comparisons can directly be made between phase diagrams of polymer only systems and those with inorganic particles. The *total* packing fraction will be larger than ϕ_P when $N_{\mathrm{inorganic}} > 0$.

Polymer stretching is analyzed by directly computing the distance r_{e2e} between the two end beads of each polymer chain in the simulation and over many saved configurations at equilibrium. All computed r_{e2e} values are presented in a histogram where any preferred configurations are visible as peaks.

The presence of ordered phases is tested by checking for the presence of peaks in the

calculated structure factor

$$S(\vec{q}) = C_0 \left\langle \left| \sum_{j=1}^{N_{\text{beads}}} e^{i\vec{q}\cdot\vec{r}_j} \right|^2 \right\rangle , \tag{7.7}$$

where $\vec{r}_j$ is the real-space position of the j'th bead in the sum, $\vec{q}$ is a discrete 3 dimensional reciprocal space vector where each component is a multiple of $\frac{2\pi}{L}$ to obey the periodic boundary conditions, C_0 is chosen so that $S(0) = 1$ and the angle brackets $\langle\rangle$ denote a time average. N_{beads} is the number of beads over which the structure factor is being calculated, typically either the hydrophobic beads B or the inorganic particles I. Calculating this sum directly is computationally expensive, so a Fast Fourier Transform technique [79] is used allowing nearly instant analysis of any simulation run.

Plots of density along a line are prepared by first continuing the simulation of the phase in question and dumping snapshots every 10,000 time steps. Then the simulation box is split into 1σ wide cubes and the packing fraction of each bead type in each cube is computed by averaging. Finally, the packing fraction along a given line can be plotted via cubic interpolation of the densities from this grid.

7.4 Phase diagram of pure systems

A previous paper discusses the phases of the pure system [64], but the details need to be elaborated here further in order to understand the effect of the inorganic components. In order to rationalize the phase diagrams obtained, it is convenient to classify polymer architectures as "hairy" when $n \gg m$, and "cross-cut" when $n \ll m$. Also, while this study focuses on functionalized triblocks, the results should also qualitatively apply to functionalized diblocks $CA_nB_{m/2}$ at the same concentration ϕ_{P}. The only difference in free energy between the two systems is the entropy loss associated with the linking of the two diblocks to form a triblock, which is presumably small.

7.4.1 Systems without inorganic components

The B block in the polymer is hydrophobic and drives the system to form micelles. The micelle *core* consists of hydrophobic monomers in a radius R_c. The *corona* consists of hy-

drophilic monomers and solvent in a thickness R_h. In the dilute regime where micelles are almost non-interacting and the hydrophilic blocks are in good solvent conditions, micelles are spherical with a finite aggregation number N_{agg}. The aggregation number depends on a balance of stretching, several free energies, and the interfacial tension [77], the details of which do not enter quantitatively into the discussion here. For increasing concentration, micelles interact with each other, eventually forming a gel.

7.4.1.1 Cross cut architecture

When n is much less than m, $R_c/R_h \gg 1$ and the corona will be insensitive to the curvature of the core. The A blocks will form a planar brush with grafting density σ_{brush} (not to be confused with the length scale σ) independent of R_c. As polymer concentration is increased, micelles interact with the potential of a compressed planar brush [133]. Most relevant is that the free energy of interaction grows linearly with the contact area among micelles. Therefore, the minimum free energy is attained by the phase that minimizes the contact area, the lamellar. That is to say, under general conditions cross-cut micelles can only exhibit lamellar phases at high concentrations.

7.4.1.2 Hairy architectures

Hairy architectures are defined by $n/m \gg 1$ which implies that $R_c/R_h \ll 1$. In this case, the stretching free energy of the hydrophilic A blocks is sensitive to the curvature of the core. It grows logarithmically with n for a spherical micelle [134], linearly with n for a planar one [133], and with an intermediate power ($n^\alpha, \alpha < 1$) for a cylindrical one. So in the hairy case, the penalty for stretching hydrophilic blocks to form cylinders or lamellar phases is high and only spherical micelles are possible. Therefore, disordered phases of spherical micelles and cubic micellar crystals are generally favored, although symmetries other than cubic are possible.

7.4.1.3 The intermediate case

The intermediate case, where $2n \sim m$, is more difficult to analyze. In the more general case, spherical micelles will remain stable beyond the gelation point and micellar crystals may form. As polymer concentration is increased, micelles will interact more strongly. By the same arguments as discussed for the cross-cut case, the free energy contribution minimizing the contact area will directly compete with the stretching of the A blocks, and a hexagonal phase of cylindrical micelles results. Further increasing the concentration leads to the lamellar phase. Depending on the actual values of n and m either the cubic (for $n \lesssim m/2$) or the lamellar phase (for $n \gtrsim m/2$) may be absent. Extensive phase diagrams for Pluronic systems support the entire previous analysis [46].

Extensive literature in melts [135] finds bicontinuous phases between the hexagonal and lamellar. Typically the phase is a gyroid, but others are possible [128]. It is therefore very reasonable to expect bicontinuous phases between hexagonal and lamellar phases in solution as well. Indeed, gyroid phases are occasionally found in Pluronics [136], but their domain of stability is very narrow.

7.5 Results

Previous studies have looked at a single polymer architecture ($CA_5B_7A_5C$) and investigated the phase diagram of these polymers combined with inorganic particles as a function of both polymer and inorganic particle concentration, inorganic particle size [13] and affinity [12]. This work expands the horizon to a more general polymer architecture $CA_nB_mA_nC$.

However, as the number of hydrophobic beads only determines the kinetic temperature where micellization occurs [64], the value of m will be fixed at 7. There is some loss of generality by making this choice even for pure systems without inorganic components, as the phase diagram is not a function only of the relative fraction of B to A monomers (see ref. [137] for a counterexample). But this nuance only changes the phase diagram in relatively minor ways that can be ignored for the purposes of this study.

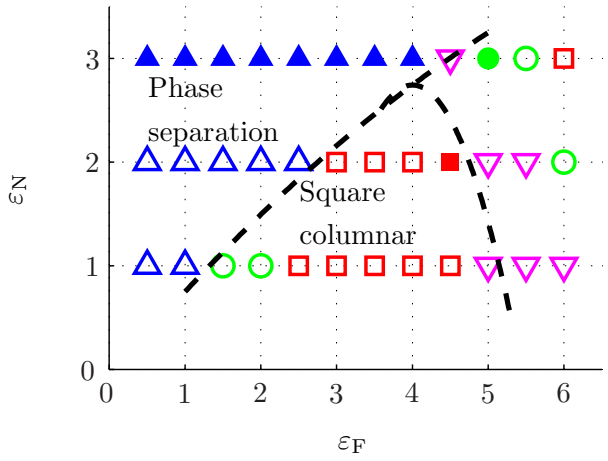

Figure 7.2 Phase diagram obtained from the simulation runs of $CA_5B_7A_5C$ at $\phi_P = 0.20$. Symbols indicate points where simulations are carried out and are filled where the inorganic particles form a crystalline phase. For a legend, see Figure 7.3

7.5.1 General procedures for simulations with inorganic particles

To begin the simulations, $N_{poly} = 800$ polymers are initially randomly placed in the simulation box. Following the work in ref. [12], two inorganic particles are added for each polymer end bead resulting in $N_{inorganic} = 3200\ I$ beads. Simulations at each point in the phase diagram initially start with k_BT/ε set to the maximum of ε_F, ε_N and 3.0. As time progresses, the system is slowly cooled over 20 million time steps to the final kinetic temperature of $k_BT/\varepsilon = 1.0$ and run for an additional 20 million time steps. Trajectories of the entire simulation run are analyzed visually to identify the equilibrium phases. Additional quantitative analyses, including structure factor calculations, diffusion coefficient determination, etc..., are performed on specific runs when needed. When needed, runs are performed from several different initial conditions or at different system sizes to asses possible metastable states or problems related to finite-size effects.

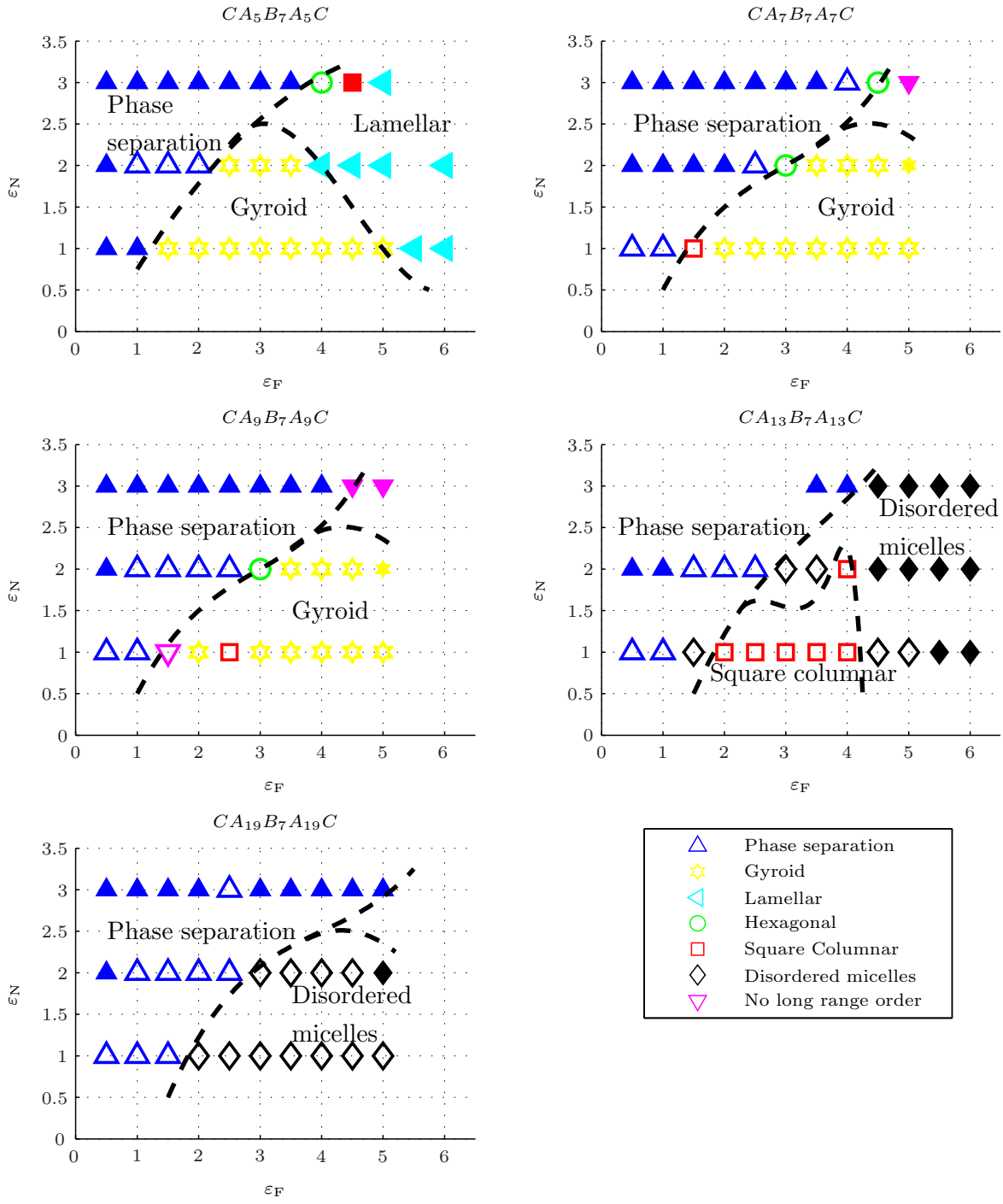

Figure 7.3 Phase diagrams obtained from the simulation runs of $CA_nB_mA_nC$ at $\phi_P = 0.25$. Symbols are filled where the inorganic particles form a crystalline phase.

7.5.2 Phase diagrams of $CA_nB_7A_nC$

Figures 7.2 and 7.3 show the phase diagrams of $CA_nB_7A_nC$ obtained via simulation at $\phi_P = 0.20$ and 0.25 respectively. At $\phi_P = 0.20$ there is a wide region where the polymers and inorganic particles phase separate (roughly when $\varepsilon_N \gtrsim \varepsilon_F$). The square columnar phase dominates the rest of the phase diagram while the hexagonal phase only shows up at a few scattered points. At this concentration the polymer architecture does not significantly change the phase diagram up to $n = 9$.

At $\phi_P = 0.25$ the large region of phase separation still exists for $CA_5B_7A_5C$. However, the square columnar phase is replaced by an even larger gyroid phase. Increasing $\varepsilon_F \gtrsim 5.0$ drives the system into a lamellar phase where the inorganic particles are in a crystalline state. As n is increased to 7 and then 9, the lamellar phase disappears with the gyroid phase becoming dominant. A further increase of n to 13 results in a small square columnar phase along with a large region of disordered micelles. Finally, at $n = 20$ the square columnar phase disappears, replaced completely by disordered micelles.

Figure 7.10 plots densities of the beads in the simulation along lines passing through features in the various phases. Without the presence of the I beads, there is a significant amount of interdigitation between the hydrophilic blocks of polymers from neighboring micelles. When added, the I beads colocate with the functionalized ends C and place themselves between these polymers forcing parts that were interdigitated to separate.

7.6 Structural properties of the phases

7.6.1 Hexagonal phase

At a few discrete points on the phase diagram of those polymer with $n = 5$, 7, and 9 both at $\phi_P = 0.20$ and 0.25, a hexagonal phase is found. Hydrophobic B beads form cylindrical cores which pack in a hexagonal pattern and are surrounded by the hydrophilic A beads. The inorganic I beads form a honeycomb structure flowing between the cylinders where the polymer end beads C are also found. Figure 7.4 is a snapshot of this equilibrium phase. While this

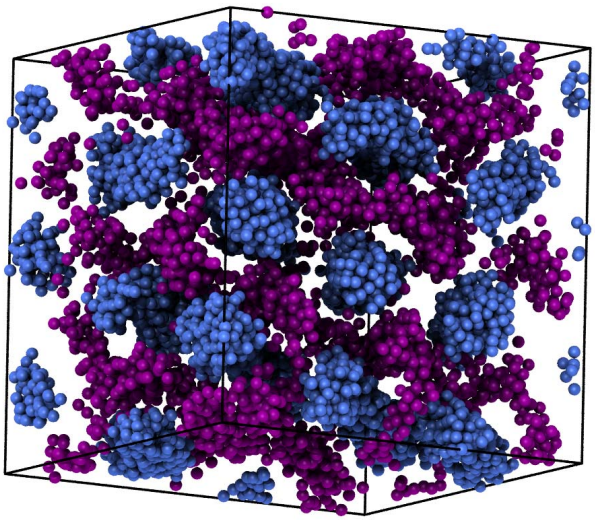

Figure 7.4 Snapshot of the hexagonal phase obtained in the $CA_5B_7A_5C$ simulation run at $\phi_P = 0.20$, $\varepsilon_N = 1.0$, and $\varepsilon_F = 2.0$. B beads are blue and I beads are purple. A and C beads area not shown to improve clarity. All snapshots in this work are created in VMD [19] and raytraced with Tachyon [138]

.

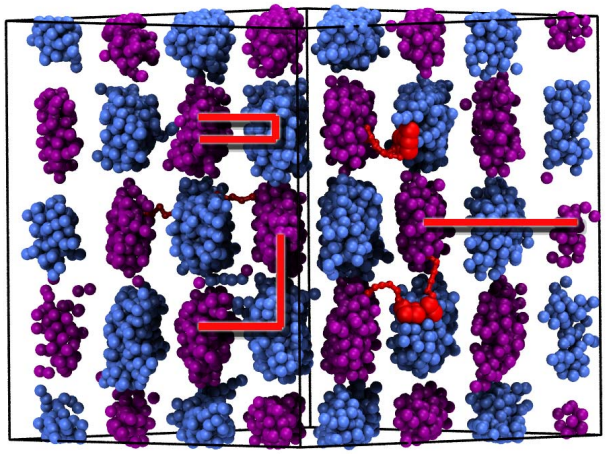

Figure 7.5 Snapshot of the square columnar phase obtained in the $CA_5B_7A_5C$ simulation run at $\phi_P = 0.20$, $\varepsilon_N = 1.0$, and $\varepsilon_F = 4.0$. Coloration is the same as in Figure 7.4, except that U, L, and fully stretched polymers are highlighted in red to demonstrate these preferred orientations.

phase appears either very narrow or even metastable at the parameters chosen for this work, it is likely to be more stable with a higher concentration of I beads as is found in ref. [13].

7.6.2 Square columnar phase

A wide region of the $CA_5B_7A_5C$ phase diagram at $\phi_P = 0.20$ is covered by the square columnar phase [13]. It also appears at $\phi_P = 0.25$ with the polymer $CA_{13}B_7A_{13}C$. Hydrophobic beads B arrange in cylindrical micelles which order on a *square* lattice while the inorganic particles I aggregate into cylindrical collections on an offset square lattice. Interestingly, the micelles formed here take on a rectangular (as opposed to a circular or square) cross-section as can be seen in Figure 7.5. The elongation of the cross-section increases as ε_F and ε_N increase.

The polymer end to end distance distribution, plotted in Figure 7.6, has a distinct multi-model shape that indicates the polymers tend to sit in preferred orientations. By highlighting

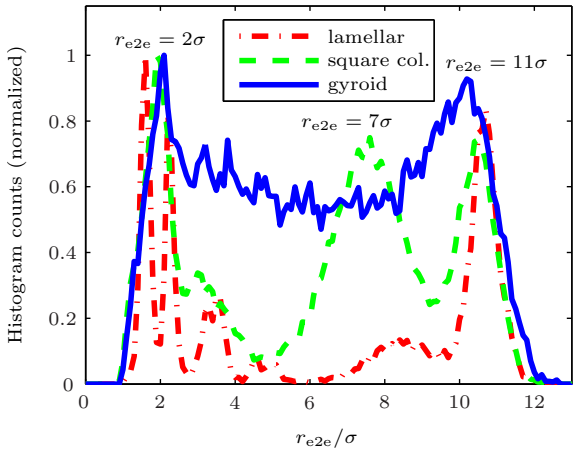

Figure 7.6 Polymer end to end distance distributions obtained from various phases found in the the $CA_5B_7A_5C$ simulation runs.

and tracking the motions of individual polymers in the movie of the simulation, these orientations are identified and correlated with the peaks. First, the peak near $r_{e2e} = 2\sigma$ in the distribution corresponds to polymers in a U configuration where the two end beads are connected to the same inorganic particle cylinder. Second, the peak near $r_{e2e} = 7\sigma$ matches up with polymers in a L configuration where the hydrophobic B block is in one of the hydrophobic cylinders with the two halves of the polymer each stretching out to a different inorganic cylinder making a L shape. Lastly, the peak at $r_{e2e} = 11\sigma$ corresponds to polymers that have fully stretched into a straight line with end blocks buried in opposite inorganic particle cylinders, as shown in Figure 7.5.

7.6.3 Gyroid phase

At $\phi_P = 0.25$, a wide region of the phase diagram is covered by the gyroid phase. This phase is characterized by the hydrophobic beads forming one interconnected network of tubes across the simulation box. The inorganic particles form another interconnected network of tubes interwoven within the first, and the two networks do not touch. Figure 7.7 shows the network of hydrophobic beads. Structure factors are calculated and the peak positions inspected, verifying that this configuration is indeed a gyroid.

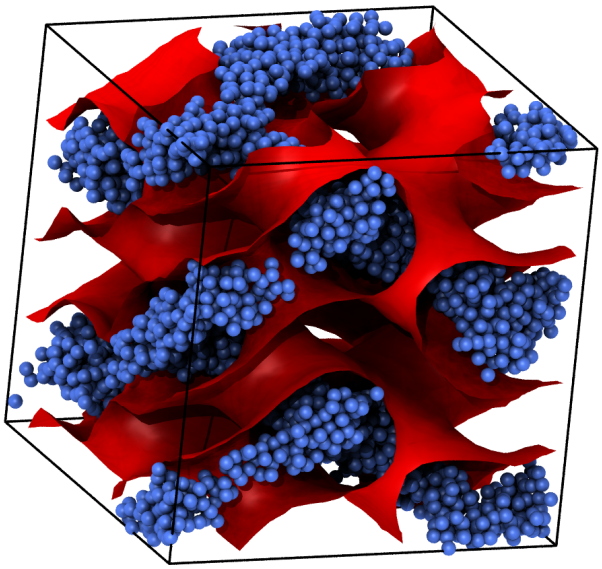

Figure 7.7 Snapshot of the gyroid phase obtained in the $CA_5B_7A_5C$ simulation run at $\phi_P = 0.25$, $\varepsilon_N = 1.0$, and $\varepsilon_F = 2.5$. B beads are blue. A, C and I beads area not shown to improve clarity.

As with the square columnar phase, polymers are found in preferred orientations. The end to end distance distribution in Figure 7.6 includes only 2 peaks corresponding to the U and fully stretched conformations.

7.6.4 Disordered micelle phase

The phase diagram of $CA_{19}B_7A_{19}C$ is dominated by a region of disordered micelles. A structure factor analysis is performed on each and every simulation run in this phase to look for the possibility of cubic micellar crystals [64]. Peaks indicating an ordered structure are only observed in a single run, the one at $\phi_P = 0.25$, $\varepsilon_N = 2.0$, $\varepsilon_F = 3.5$. A lattice of micelles is present offset from a lattice of spherical aggregates of inorganic particles, similar to the cubic phase found in ref [13]. Due to the extreme amount of computer time that would be required [64], no attempt is made to ascertain the boundaries of this ordered cubic phase or it's reproducibility.

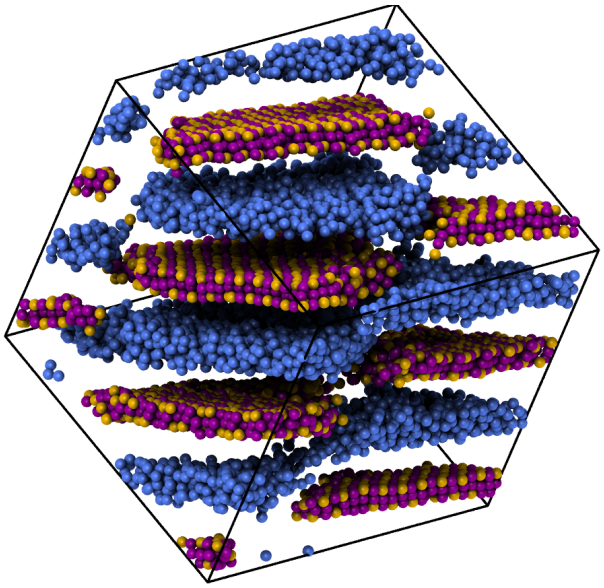

Figure 7.8 Snapshot of the lamellar phase obtained in the $CA_5B_7A_5C$ simulation run at $\phi_P = 0.25$, $\varepsilon_N = 2.0$, and $\varepsilon_F = 4.0$. B beads are blue, I beads are purple and C beads are orange. A beads area not shown to improve clarity.

7.6.5 Lamellar phase with crystallized inorganic component

The lamellar phase shows up at $\phi_P = 0.25$ when the value of of ε_F is high and the length of the hydrophilic block is short ($n = 5$). It is characterized by large planes of hydrophobic beads alternating with planes of inorganic ones. In combination with the polymer end blocks C, the I beads display additional in-plane crystalline order. The crystal alternates beads $ICICIC$ as on a checkerboard in order to achieve the lowest potential energy U_{IC}.

The end to end distance of the polymers in this phase is plotted in Figure 7.6. There are two dominant peaks, one at small r_{e2e} corresponding to the U conformation and one at $r_{e2e} = 11\sigma$ corresponding to the fully stretched conformation. A number of smaller peaks is also present at small r_{e2e}. These correspond to the two end beads of the polymer being separated by a varying number of beads in the crystal plate.

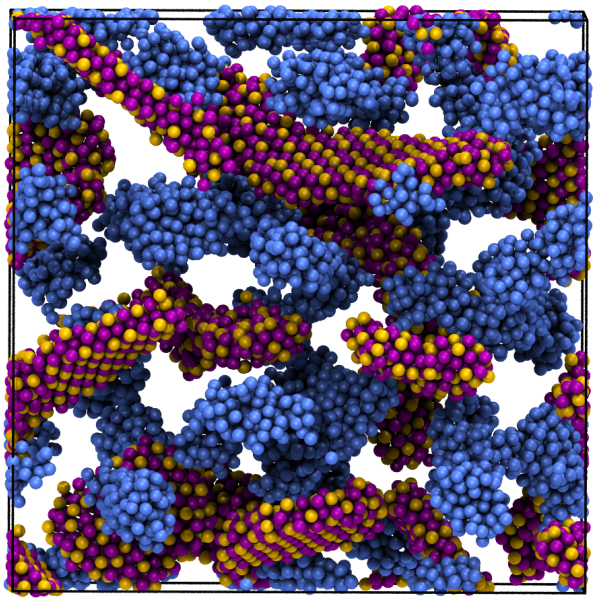

Figure 7.9 Snapshot of the disordered micelle phase obtained in the $CA_{19}B_7A_{19}C$ simulation run at $\phi_{\mathrm{P}} = 0.25$, $\varepsilon_{\mathrm{N}} = 2.0$, and $\varepsilon_{\mathrm{F}} = 5.0$. Coloration is the same as in Figure 7.8

7.6.6 Disordered micelles with inorganic platelets

A number of other points in the phase diagram show up with randomly oriented crystallized plates of inorganic particles and polymer end blocks, similar to those found in the lamellar phase. These plates are much smaller in width (10's of σ) and the overall system exhibits no long range order, as depicted in Figure 7.9.

7.7 Discussion on the origin and stability of the phases

7.7.1 Addition of inorganic components

Inorganic components introduce two characteristic energies, inter-particle attraction (ε_N) and their affinity for the functionalized end-blocks (ε_F). When both energies are below the characteristic thermal energy scale of $k_{\mathrm{B}}T$, inorganic particles behave like solvent. When particle attraction is strong ($\varepsilon_N/\varepsilon_F \gg 1$), the inorganic component inevitably phase separates from the polymeric matrix. Therefore, a successful nanocomposite is only possible when $\varepsilon_N \lesssim$

ε_F. There are two cases, categorized based on whether the inorganic component crystallizes:

- The curved regime ($\varepsilon_F/(k_\mathrm{B}T) > 1$)
- The flat regime ($\varepsilon_F/(k_\mathrm{B}T) \gg 1$)

Based on the simulation results, the boundary between the two regimes is roughly at $\varepsilon_F/(k_\mathrm{B}T) \sim 5$ (see Figure 7.3).

7.7.1.1 The curved regime

In the curved regime, inorganic particles attract each other but do not crystallize. The phases induced by the addition of inorganic particles are dependent on polymer architecture. At high concentrations, systems of cross-cut micelles form lamellar phases only. Therefore, the inorganic particles will colocate with the functionalized end blocks, resulting in ordered inorganic structures following the existing lamellar polymer matrix. While cross-cut architectures do result in a successful composite, they are restricted to lamellar phases.

For hairy micelles, the free energy required to stretch polymers will be exceedingly large, and spherical micelles will remain stable even in the presence of inorganic components. If the attraction of the inorganic components is strong enough, they may aggregate into larger entities much like large nanoparticles. Ordered phases such as the lamellar catenoid (referred to as layered hexagonal in ref. [13]) or CsCl are possible in these types of systems [13]. The phase diagram of the $CA_{19}B_7A_{19}C$ system shown in Figure 7.3 is well explained from these arguments.

The discussion on pure systems identifies that intermediate polymer architectures ($m \sim 2n$), neither cross-cut nor hairy, allow transitions between spherical, hexagonal, lamellar and bicontinuous phases controlled via polymer stretching. In the presence of the inorganic component these phases remain present (see Figure 7.3), offering a rich set of possibilities for polymer nanocomposite design.

The role of polymer stretching in this intermediate regime is revealed by the density profile analysis of the $CA_5B_7A_5C$ architecture shown in Figure 7.10. In the absence of inorganic

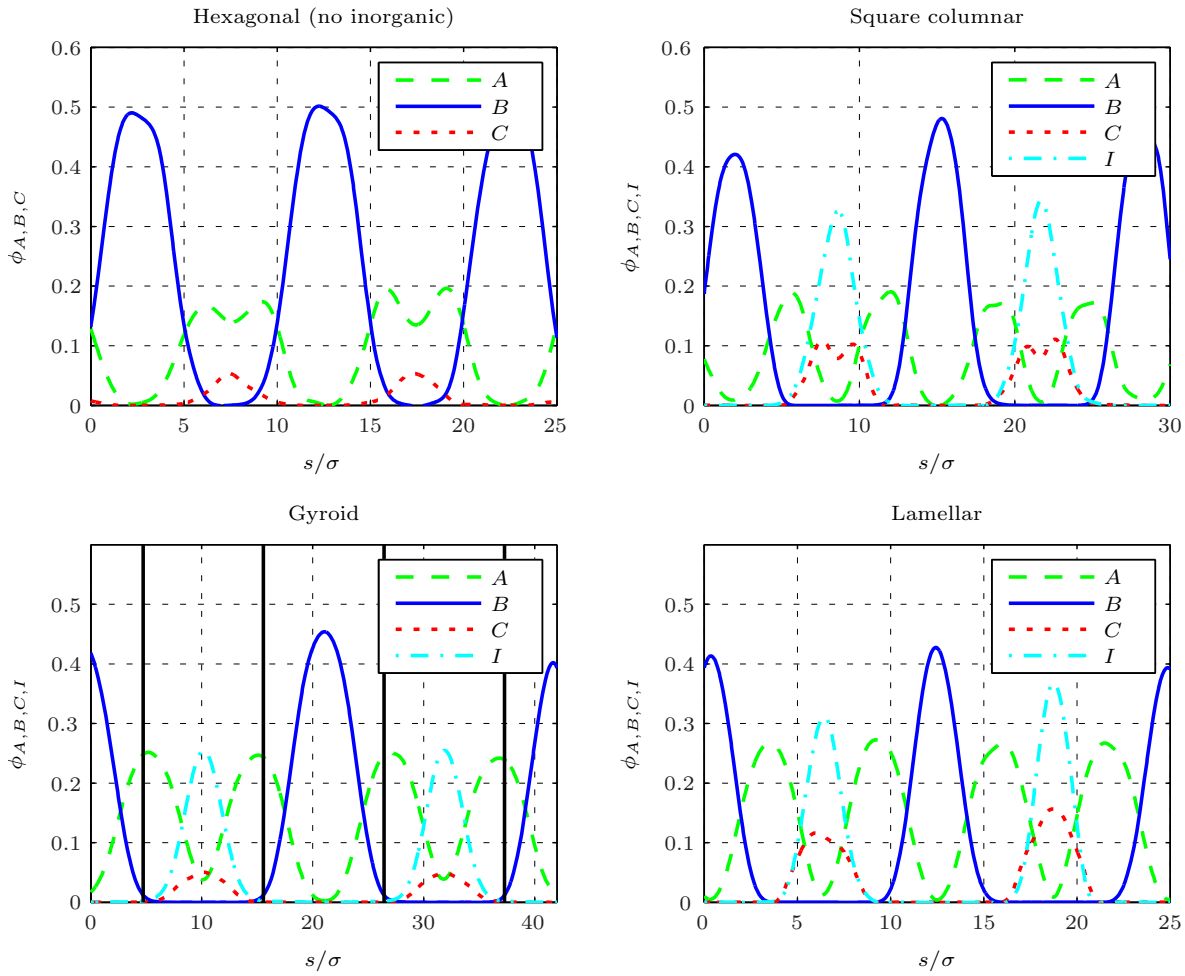

Figure 7.10 Plots of the density of polymer and inorganic beads along a given line. In the plot for the hexagonal phase, the line passes through 3 neighboring cylinders. In the square columnar phase, the line is chosen to pass diagonally from one hydrophobic cylinder through the inorganic component and on to the next nearest neighbor. The line specified for the gyroid phase passes perpendicular to the tube network and through a node. Finally, in the plot of the lamellar phase, the line chosen is perpendicular to the lamellar plates.

particles, a hexagonal phase forms. The A and C beads taken together conform to a textbook compressed brush [133]. When inorganic particles are included, they move to a location midway between neighboring hydrophobic cores. The hydrophilic A blocks form a brush and the functionalized end beads localize in the surface defined by the inorganic particles. The A blocks are more localized in space as compared with the situation in the absence of inorganic particles, so they become more stretched. Therefore, cylindrical micelles can transition to bicontinuous or lamellar morphologies.

Focusing further on the $CA_5B_7A_5C$ architecture, micelles remain cylindrical at low concentrations (see Figure 7.2) where a square columnar phase is found. The same phase is also found in systems with large nanoparticles as reported in Ref. [13]. Based on other findings in that work, the square columnar phase found here should also become unstable towards a hexagonal phase for higher inorganic particle concentrations.

Upon increasing polymer concentration, phases that favor minimum contact area at the expense of some polymer stretching should be favored by the same arguments as discussed for pure systems. Indeed, a gyroid phase is found (see Figure 7.3). Given how difficult stabilizing gyroid phases is in pure systems, it is quite remarkable how wide the domain of stability is in the presence of inorganic components.

For increasing hydrophilic length n, the stretching energy necessary to stabilize the gyroid phase becomes unfavorable, and the square columnar phase should be recovered at the same polymer concentration. Indeed, $CA_{13}B_7A_{13}C$ at $\phi_P = 0.25$ does go to a square columnar phase.

7.7.1.2 The flat regime

In the flat regime the inorganic phase crystallizes, thus favoring flat structures consisting of a square lattice where I and C beads alternate as on a checkerboard. The square lattice is preferred because a triangular lattice would induce frustration and a curved structure is higher in energy due to buckling. Therefore, only architectures that form lamellar phases are naturally compatible with the tendency of the inorganic component to induce flat structures and exhibit

long range order (see Figure 7.8). For other architectures, the tendency of the inorganic phase to favor flat structures will be frustrated by the stretching of the underlying polymer matrix. A phase of disordered micelles with dispersed small size flat crystallites results as shown in Figure 7.9 . These arguments explain the flat regime of the phase diagrams in Figure 7.3.

7.8 Conclusions

7.8.1 Summary of results

Designing polymer nanocomposites for new materials presents substantial challenges. The tendency of the inorganic components to phase separate is one of the most recognized difficulties. In many cases, the *inorganic* component is functionalized in different ways [132] so as to inhibit particle attraction. The extensive analysis presented in this paper, combined with previous results [12, 13] show that *polymer* functionalization is a viable and powerful alternative for polymer nanocomposite design.

For symmetric triblocks, this study has identified architectures with $n \sim m$ as providing the richest variety of phases with inorganic component showing different tunable morphologies. The case where $n \ll m$ (cross cut) is suitable for designing a composite where ordered planes of crystallized inorganic particles (see Figure 7.8) are required. In the opposite case, $n \gg m$, small crystallites form which may be suitable for systems where nano-sized crystallites are required to be dispersed into a polymeric matrix. Following previous discussions, all these results should also apply for functionalized diblocks of the form $CA_nB_{m/2}$.

The different composites are not only dependent on polymer architecture, but on the affinity between I and C blocks, which is parameterized by the energy scale ε_{F}. When $\varepsilon_{\mathrm{F}}/k_{\mathrm{B}}T \gtrsim 5$, the composite contains some degree of crystalline order and in some cases, mesoscopic order as well. While for $\varepsilon_{\mathrm{F}}/k_{\mathrm{B}}T \lesssim 5$, the inorganic particles are liquid-like, but the systems display a far richer degree of mesoscopic order. In addition, this analysis establishes that the polymers are always strongly stretched [86] (but not described by the strong segregation limit, as the A–B interface is not sharp) and that the balance between stretching and affinity determines the stability of the different phases observed.

7.8.2 Experimental implications

At high inorganic/polymer affinity, our results suggest a new mechanism to assist crystallization of inorganic components by using a suitable choice of functionalized polymers. This leads to a situation where the polymer end blocks bridge the inorganic components, as shown in figures 7.8 and 7.9.

At lower affinity, this study suggests a new method for creating polymer nanocomposite materials in situ, where the nanoparticles are not necessarily spherical. For example, in the square columnar phase (Figure 7.5) the inorganic components are liquid, but by further lowering temperature they might be induced to crystallize, thus forming a composite with cylindrically shaped nanoparticles.

The most widely studied polymer nanocomposite is that of a diblock (A_nB_l) polymer melt with dispersed nanoparticles. Usually, the phase diagram is presented as a function of nanoparticle selectivity determining whether the nanoparticles have an affinity for either the A or B blocks respectively [122,123,129]. Non-selective nanoparticles have no attraction for either the A or B blocks. In those studies, the original lamellar order of the pure system remains, although some quantitative parameters, such as the lamellar spacing change. There has been a recent interest in systems of "active" nanoparticles [119], where the resulting composite exhibits a different type of order than the original pure polymer system. Polymer functionalization as shown in this work is a strategy where nanoparticles are "active" by this definition. The inorganic components do not simply accommodate to the mesoscopic order of the polymer matrix in the absence of inorganic components, but actively modify the original phase so that the final mesoscopic order is dictated by both the polymer and the inorganic component. The investigation of diblock or triblock copolymer melts with functionalized polymers is a promising new direction that we expect will be the subject of future experiments.

Experimental examples of functionalized systems are presented in ref. [130]. The Pluronic polymer used is F127, $PEO_{99}PPO_{65}PEO_{99}$, an example of a hairy system. The polymer is functionalized with different groups that have affinity for calcium phosphate in an attempt to synthesize a material similar to natural bone. Detailed imaging and X-ray experiments show

that calcium phosphate crystallized, usually forming nanometer sized platelets but without mesoscopic order. This bears a distinct similarity to the phase shown in Figure 7.9. However, a direct comparison between theory and experiment becomes difficult as it is not clear to what extent the experimental results are in equilibrium nor to what extent the approximation that calcium phosphate is non-selective to Polyethylene oxide.

There has been a significant amount of theoretical and experimental work on tethered nanoparticles, nanoparticles with attached polymer chains. The functionalized polymers in this work reduce to tethered nanoparticles in the limit of 1) large affinity ε_{F} and 2) the additional restriction that only one inorganic bead can be attached to a functional group. Indeed, phase diagrams of tethered and functionalized systems show considerable similarities, such as the presence of a gyroid phase, but a detailed comparison of both systems will be left for a subsequent study. [139]

Certainly, there are other mechanisms for polymer nanocomposite design besides polymer functionalization. For example, the nanoparticle radius can be tuned to be similar to the micelle radius. Then the nanoparticles will be unable to diffuse through a cubic micellar crystal, and a low concentration of such nanoparticles might remain ordered located in the interstitial spaces of a micellar gel. In this case, nanoparticle repulsion may help stabilize such phases, as it should distribute nanoparticles more evenly during the gelation process. The effects of nanoparticle size is studied via simulation in ref. [13], and no successful nanocomposites where observed without functionalization. However, the largest nanoparticle size studied was considerably smaller than micelle radius. [140]

There are a number of parameters that this study has not addressed. Most notably the influence of inorganic concentration and effects related to inorganic particle size. These effects were analyzed systematically in a previous paper [13] showing additional phases not found in this study, such as the layered hexagonal phase. It remains for a future publication to complement the systematic study of polymer architecture with these additional parameters.

7.9 Acknowledgments

We thank Lynn Walker for many discussions. Our colleagues at Ames lab M. Akinc, M. Lamm, S. Mallapragada, J. Schmalian and K. Schmidt-Rohr are acknowledged for illuminating discussions. A.T. wants to acknowledge the Aspen Center for Physics and specially Kristen Grosse-Brauckmann for discussions. Many of the simulations presented in this work were executed on Tesla GPUs provided by NVIDIA through their professor partnership program. Sharon Glotzer provided time on the GPU computer cluster at the University of Michigan.

This work is funded by DOE through the Ames lab under Contract DE-AC02-07CH11358.

CHAPTER 8. NUMERICAL ERRORS IN MOLECULAR DYNAMICS

8.1 Introduction

Any numerical calculation performed using finite precision floating point arithmetic is subject to error [141] simply from the limited precision of the values stored. IEEE-754 single precision arithmetic, the standard implemented on most computers and NVIDIA GPUs, stores binary floating point numbers as $s \cdot m^p$ where s is a 1-bit value representing the sign of the number, m is the 23-bit mantissa and p is the 8-bit exponent. The mantissa is large enough to represent roughly 6 decimal significant digits of precision. Double precision increases the bits in the mantissa to 52 and those in the exponent to 11, making roughly 15 decimal significant digits of precision available and a wider range of numbers can be represented.

The result of this finite precision has many consequences [141] that must be taken into consideration to reduce errors when writing any numerical computer program. The most important type of error for the purposes of the following discussion is called *round-off error*. To put it simply, in finite precision floating point arithmetic $1 + \epsilon$ is evaluated to be exactly 1 for all $\epsilon < \epsilon_m$ where ϵ_m is called the *machine epsilon*. One can see this more clearly on numbers with larger exponents; Adding $1 \cdot 10^7 + 1$ in base 10 with only a finite 6 significant figures leaves the 1 off the end and the resulting value is still $1 \cdot 10^7$.

Errors resulting from finite numerical precision are not the whole story, however. Computer simulations are typically performed as approximate solutions to a problem when no analytical one exists [16]. Errors specific to the problem and type of approximation used are also very important, and often times much greater than any errors introduced from performing calculations in finite precision arithmetic. Such errors are sometimes called *truncation errors* because they result from the truncation of an infinite series. [142]

8.2 Sources of numerical errors in molecular dynamics

Every approximation made in the derivation of molecular dynamics is a potential source of error. First and foremost, molecular dynamics is an iterative solution of $3 \cdot N$ coupled ordinary differential equations, where N is the number of particles in the simulation. The solution is evaluated only at discrete points in time $t_n \cdot \delta t$ where t_n is an integer and δt is the step size. Discretizing a continuous solution to an ODE introduces an approximation with error in $O(\delta t^m)$ where m is the *order* of the integrator being used [16]. Second, most molecular dynamics simulations (including all those presented in this thesis) have an additional approximation that the potential energy $V(r)$ between pairs of particles is cutoff to 0 when $r \geq r_{\text{cut}}$. Additional errors can be introduced by this.

Molecular dynamics is pretty well-behaved as far as errors from finite precision arithmetic go. There are few areas where serious problems, such as cancellation errors, occur except in extremely large simulations of hundreds of millions of particles. The biggest problems that show up in small and medium sized simulations are round-off errors resulting from an improper choice of the step size δt. The simple Euler integrator [16] illustrates this problem simply. In it, the velocity is moved forward a single time step by the calculation $\vec{v}(t+\delta t) = \vec{v}(t) + \vec{a}(t) \cdot \delta t$. This is identical to the form of round-off errors discussed in section 8.1, with $\vec{a}(t) \cdot \delta t$ playing the role of ϵ. Thus, it is clear that if δt is chosen *too small*, round-off errors will result. However, with the truncation error proportional to δt^n, choosing *too large* of a step size is also problematic. An actual evaluation of the error at various step sizes must be performed in order to choose one with an acceptable margin of error (see section 8.4).

8.3 Minimizing numerical errors in molecular dynamics

The best way to minimize round-off errors is to use the highest precision calculations that the hardware makes available. Typically in standard CPU applications, this means using double precision at an insignificant performance loss vs. the same calculation in single precision. On current NVIDIA GPUs, however, the performance loss for switching completely to double precision can be as much as a *factor of* 2–8, which would negate many of the great performance

gains obtained in chapter 6.

To get the best of both worlds, some algorithms currently being implemented on GPUs are using *multi-precision* methods [143] where single precision is used in most of the algorithm for performance reasons and double precision is only used where absolutely necessary. Such applications result in the same error levels as their fully double precision counterparts, but without any significant performance loss. A detailed analysis of errors in single vs double precision molecular dynamics is performed in section 8.4 and it is hoped that the results will aid in future work developing a multi-precision implementation on the GPU.

To reduce the errors introduced by cutting off the potential, some existing molecular dynamics applications, such as NAMD [23] and GROMACS [101], introduce a *smoothing* function to smoothly vary both $V(r)$ and the force to 0 at $r = r_{\text{cut}}$. Specifically, NAMD [23] uses the XPLOR smoothing function that modifies the Lennard-Jones potential function to be $S(r) \cdot V_{\text{LJ}}(r)$ with

$$S(r) = \begin{cases} 1 & r < r_{\text{on}} \quad (8.1) \\ \frac{(r_{\text{cut}}^2 - r^2)^2 \cdot (r_{\text{cut}}^2 + 2r^2 - 3r_{\text{on}}^2)}{(r_{\text{cut}}^2 - r_{\text{on}}^2)^3} & r_{\text{on}} \leq r \leq r_{\text{cut}} \quad (8.2) \\ 0 & r > r_{\text{cut}} \quad (8.3) \end{cases}$$

. Where $V_{\text{LJ}}(r)$ is the original Lennard-Jones potential. When r_{on} is chosen to be large, say $2/3 * r_{\text{cut}}$, then the effect on the overall shape of the potential is minimal. The benefit is that the resulting $V(r)$ has C2 continuity with both the potential and force going to 0 smoothly at $r = r_{\text{cut}}$. No in-depth analysis could be found in the literature quantifying the benefits of this somewhat expensive smoothing operation, so one is performed here in section 8.4

8.4 Quantifying errors in molecular dynamics

8.4.1 Errors over short runs

Ref. [15] suggests that errors in molecular dynamics can be quantified by performing a series of relatively short simulations and examining the fluctuation of quantities that should be conserved. To perform this evaluation, a configuration of $N = 32000$ particles is first equilibrated

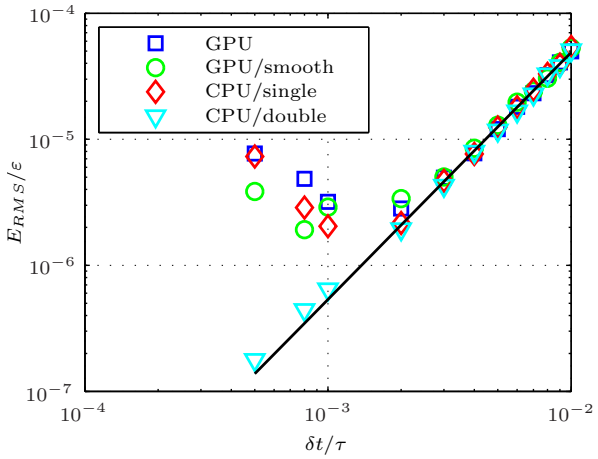

Figure 8.1 RMS error in energy vs. step size for various configurations. The *GPU* runs are performed on a NVIDIA 8800 GTX GPU with the shifted Lennard-Jones potential. *GPU/smooth* is the same, but the potential is modified with the smoothing function. *CPU/single* and *CPU/double* are runs of identical code on the CPU, but compiled with single and double precision arithmetic respectively

in a NVT ensemble with $T^* = 1.0$. Then, starting from this same initial configuration each time, NVE dynamics are continued for a time of 50τ (for an overview of units, see section 3.6) with varying step sizes δt and with the software configured in various ways. All of these simulations are performed with $r_{\text{cut}} = 3.0\sigma$. For each run, the total system energy $E_{\text{tot}}(t_j)$ is calculated every 10 time steps for a total of M samples. A root mean square fluctuation of the energy per particle is then calculated by

$$E_{\text{RMS}} = \left[\frac{1}{M} \sum_{j=1}^{M} \left(E_{\text{tot}}(t_j) - \left(\frac{1}{M} \sum_{k=1}^{M} E_{\text{tot}}(t_k) \right) \right)^2 \right]^{1/2} /N$$

.

Results are shown in Figure 8.1. In the entire region where $\delta t/\tau > 0.001$ the magnitude of the error is virtually identical for all software configurations. This shows that there is nothing significantly "inferior" about the non IEEE-754 compliant single precision arithmetic on the GPU. It also shows that neither the double precision nor the use of the smoothing function result in smaller errors of energy conservation. For values of $\delta t \leq 0.001$, the double and single

precision calculations split with the single precision experiencing an upturn in the error. This is a classic example of round-off error in action [16], as previously discussed. The double precision runs can continue with a lower error at a lower δt because the value of machine epsilon is smaller for double precision. There should eventually be an upturn of the double precision runs, too, but a significant amount of computation time is needed to run simulations at that small of a step size so they are not performed.

We can learn a little about the source of the errors from Figure 8.1. The black best fit line plotted has a slope of 2 on the log-log plot providing a good indication that the errors result from the approximation made in the velocity-verlet integration algorithm used to implement NVE simulations in this software (which has error proportional to $O(\delta t^2)$ [15]).

Step sizes larger than a δt of 0.01 were not stable. Such a large step size can result in two particles moving very close together from one step to the next. Now at a very small r, the force from the hard core repulsion in the Lennard-Jones potential becomes extremely high, sending the particle an even farther distance on the next step where it is likely to end up very close to yet another particle. After a few repeated steps like this, eventually something bad, like a divide by 0 happens and the whole calculation becomes garbage. The key parameter that causes this unstable behavior is the distance any given particle moves in a time step, which is approximately $v(t) \cdot \delta t$. Thus, systems with a higher temperature may need to be run at a smaller time step to keep this distance small so that the entire system remains stable.

While this intuitive description of instability is what occurs in practice, a rigorous analysis determining the stability of any numerical integrator can be performed [142]. In short, instabilities occur from the propagation of errors from one time step to the next in such a way that the calculated solution deviates from the real solution exponentially in time. A quantifiable criterion exists (related to the step size δt) that provides a hard line separating stable solutions from unstable ones, as seen in the simulations above where the simulations were unstable for $\delta t > 0.01$.

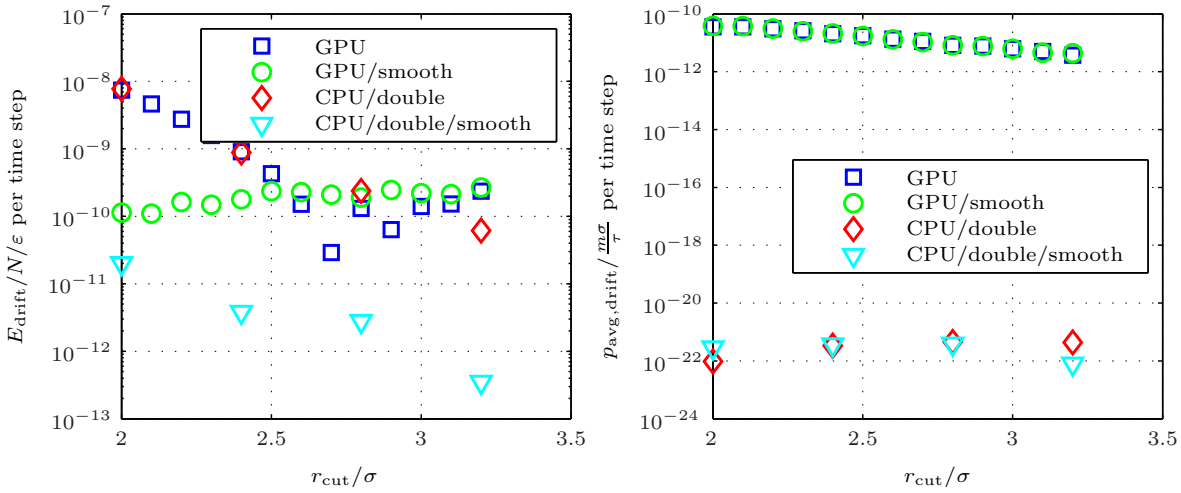

Figure 8.2 (Left) Drift of energy vs. r_{cut} for various configurations. (Right) Drift of system momentum vs. r_{cut} for various configurations. The *GPU* runs are performed on an NVIDIA 8800 GTX GPU with the shifted Lennard-Jones potential. *GPU/smooth* is the same, but the potential is modified with the smoothing function. *CPU/double* is run using double precision and shifted potentials on the CPU, and *CPU/double/smooth* is the same with the smoothing function applied

8.4.2 Errors over long runs

When it comes to predicting phase diagrams of polymer systems, or any of the other applications of molecular dynamics, 50τ is only a very brief simulation. Simulations lasting for $5 \cdot 10^5\tau$ or even longer are not uncommon. To see how numerical errors result in the drift of conserved quantities over these long time periods, initial conditions are prepared as in the previous section but this time the continuing runs are performed at a fixed $\delta t = 0.005$, for a total time of 25000τ and varying values of r_{cut}. Total system energy and momentum are both examined for any amount of drift over time by performing best line fits to the quantities vs. time and plotting the calculated slopes. Results are in Figure 8.2.

For the shifted but not smoothed potentials, the energy drift is significantly higher for smaller values of r_{cut}. This is expected as the magnitude of the discontinuity in the force grows as r_{cut} is decreased (and is still kept above the minimum of the potential at $r = 2^{1/6}\sigma$).

Including the smoothing function on the single precision GPU runs reduces the energy drift at small r_{cut}, but provides no additional benefit at larger (and more reasonable) values of r_{cut} when compared to the GPU runs with shifted potentials and even the CPU runs in full double precision with shifted potentials. Other cutoff potentials with a larger force magnitudes at $r = r_{\text{cut}}$, such as those needed in certain lipid bilayers [144], should benefit more from this smoothing than Lennard-Jones.

What is surprising are the results from the CPU double precision runs with smoothed potentials. In these, the energy drift is reduced by several orders of magnitude. In particular, the drift is so small for the run at $r_{\text{cut}} = 3.2\sigma$ that the calculated RMS fluctuation for this run is *the same* as the RMS calculated from the comparable 500 times shorter runs performed in subsection 8.4.1. Clearly, some combination of double precision integration and in the force calculation results in superior energy conservation over long runs. This isn't to say that the energy conservation of the single precision calculations is bad. In fact, it is small enough that all of the polymer model simulations performed on the GPU in this thesis run without problems and validate against all cross-checks performed on the CPU in double precision using a different software package (LAMMPS [22]). Only a few specialized uses of MD, such as perhaps free energy calculations which requires a very precise calculation of the total energy, might not be possible with drift induced by the reduced precision.

Momentum conservation is a different story. Calculations in single precision result in a *significant* drift in the entire system momentum over long runs. The drift rate is more than 10 orders of magnitude higher than the same calculations in double precision. The causes of this drift are not obvious and will be looked into further in the future. Fortunately, there is a simple and effective way to work around the problem. Every so often, say 500τ, calculate the average momentum of the system and subtract if from each particle. This prevents the drift from reaching a stage where it is so large that simulation results are changed from what they should be. Additionally, the total energy energy removed from the system by this operation is still well within the existing RMS fluctuation of the energy per particle: $\frac{p^2}{2m} = (500\tau/0.005\tau/\text{step} * 10^{-10}\frac{m\sigma}{\tau}/\text{step})^2/(2m) = 5 \cdot 10^{-11}\varepsilon$.

8.5 Conclusions

Errors resulting from finite precision arithmetic and approximations in the molecular dynamics algorithms are present in every simulation run. It is important to understand what these errors are and know an estimation of their magnitude so there are clear expectations of the precision when beginning any simulation. Figures 8.1 and 8.2 are the main results presenting this information. Most importantly in the context of the polymer simulations presented in this thesis, an appropriate step size can be chosen from Figure 8.1 that is stable and within an acceptable error margin, but not so large that calculations become unstable and result in divide by 0 problems (note that 0.005 is used in most of the work presented here). Other types of errors found, such as the energy drift, do not significantly effect the results of the polymer simulations.

Fortunately in this new era of GPU computing, errors when molecular dynamics is performed on GPUs are not significantly higher than corresponding simulations done in double precision on the CPU (or can be worked around in the case of the momentum problem). Hopefully the analysis presented here will be only the first step in a future work expanding the GPU computation with multi-precision arithmetic, bringing the error magnitudes down so that GPUs can be an effective tool even for those in the broader community performing specialized molecular dynamics simulations that really need extremely small errors.

CHAPTER 9. CONCLUSION

9.1 Polymers

Systems of polymers in solution exhibit a wide variety of phases. This thesis studies many of these phases in great detail from a theoretical perspective using molecular dynamics simulations. First, a number of new advances are made in technique, such as the parameterization of solvent quality in chapter 4 and the equilibration method for cubic morphologies in chapter 5. These new techniques expand the applicability of molecular dynamics to the study of a wider range of polymer systems. With the solvent quality parameterization, the temperature dependence on hydrophobicity that is present in many real monomers can now be included in coarse-grained simulations. It has already been adapted for Monte Carlo simulations and used by others in the field [145]. The exhaustive analysis of micellar crystals having cubic symmetries in chapter 5 results in an efficient and general technique that can be applied to any future simulations involving similar phases. Such phases occur often in polymer systems, so it should prove useful to much of the community studying polymers with MD.

However, the research goal set out in the introduction was not only the creation of new techniques. A solid understanding of the behavior of polymers in ordered phases is also gained from chapters 4 and 5. Quantities from the dynamics of the phase formation to detailed calculations of the density fluctuations, micelle properties, and many more all come together in the big picture and help provide a simple intuitive description of these phases.

Much of this work on pure polymer systems is a critical step for the subsequent study of polymer-inorganic composite materials completed in chapter 7. This detailed study lays out how challenging it can be to form a successful composite material. Without some affinity between the polymer and the inorganic particles, successful composites do not form as the

inorganic component either diffuses freely or phase separates from the polymer. Functionalization is introduced as a viable strategy to circumvent this problem. A wide region in parameter space is analyzed, detailing the effects of the strength of the affinity and polymer architecture. It is hoped that this detailed study will be useful as a roadmap to guide future experiments towards developing the new generation of polymer nanocomposite materials.

9.2 GPU Computing

The development of data-parallel algorithms implementing molecular dynamics on the graphics card in chapter 6 is a another major accomplishment of this thesis that impacts a large community of MD users in many fields of science. This thesis is being written two years after the initial work began on the GPU computing project. Since then, it has evolved into HOOMD (Highly Optimized Object-oriented Molecular Dynamics), an open source software project available to everyone online. It is currently being used around the world, by research groups at the University of Illinois at Urbana-Champaign, the University of Pennsylvania, the University Michigan, the University of Virginia, the Naval Research Laboratory, and many more. The drug company Pfizer has even made a few inquires about it. While HOOMD is already a useful tool for many, it has a bright future ahead as I will be continuing development during the next step in my professional career.

Also since the paper was published, NVIDIA has released a new generation of GPUs that run HOOMD 50% faster than their predecessors. With yet another new generation on the horizon, significantly faster performance will soon be obtainable again with just a cheap hardware upgrade. Combined with the new hardware, a number of algorithmic performance improvements have also been made since the paper was published. The current fastest NVIDIA GPU running the latest development version of HOOMD is now roughly *63* times faster than a single CPU core (compared to a current generation Opteron 2356). All ideas for optimization have not even been fully explored yet, and there are plans currently for at least another 10–20% performance improvement.

We are now in a new era of accelerated computing that enables bigger, faster, and cheaper

simulations than ever before. The last point has perhaps been underemphasized. As of the writing of this thesis, a mere $350 can buy a graphics card capable of running HOOMD at roughly the same level of performance as a cluster of eight dual quad-core machines connected with a fast (and expensive) network. With the performance of GPU outpacing their CPU counterparts, it seems inevitable that GPU computing is the future of scientific computing.

BIBLIOGRAPHY

[1] M. Rubinstein and R. Colby. *Polymer Physics*. Oxford University Press (2003).

[2] M. R. Bockstaller, R. A. Mickiewicz, and E. L. Thomas. *Advanced Materials*, **17** p. 1331 (2005).

[3] J. A. Anderson, C. D. Lorenz, and A. Travesset. *Journal of Computational Physics*, **227** pp. 5342–5359 (2008).

[4] R. Belleman, J. Bedorf, and S. P. Zwart. 'High performance direct gravitational N-body simulations on graphics processing units – II: An implementation in CUDA.' (2007). Accepted for publication, New Astronomy.

[5] H.-Y. Schive, C.-H. Chien, S.-K. Wong, Y.-C. Tsai, and T. Chiueh. 'Graphic-card cluster for astrophysics (GraCCA) – performance tests.' (2007). Submitted for publication, New Astronomy.

[6] J. E. Stone, J. C. Phillips, P. L. Freddolino, D. J. Hardy, L. G. Trabuco, and K. Schulten. *J. Comp. Chem.*, **28** pp. 2618–2640 (2007).

[7] J. A. van Meel, A. Arnold, D. Frenkel, S. F. P. Zwart, and R. G. Belleman. 'Harvesting graphics power for MD simulations.'

[8] B. Anderson, S. Cox, P. Bloom, V. Sheares, and S. Mallapragada. *Macromolecules*, **36** pp. 1670–1676 (2003).

[9] M. Determan, J. Cox, S. Seifert, P. Thiyagarajan, and S. Mallapragada. *Polymer*, **46** pp. 6933–6946 (2005).

[10] M. Determan, L. Guo, P. Thiyagarajan, and S. Mallapragada. *Langmuir*, **22** pp. 1469–1473 (2006).

[11] V. Ortiz, S. O. Nielsen, M. L. Klein, and D. E. Discher. *Journal of Polymer Science Part B-Polymer Physics*, **44** pp. 1907–1918 (2006).

[12] C. D. Knorowski, J. A. Anderson, and A. Travesset. *The Journal of Chemical Physics*, **128** p. 164903 (2008).

[13] R. Sknepnek, J. A. Anderson, M. H. Lamm, J. Schmalian, and A. Travesset. *ACS Nano*, **2** pp. 1259–1265 (2008).

[14] P. De Gennes. *Scaling Concepts in Polymer Physics*. Cornell University Press, Ithaca, New York (1979).

[15] M. Allen and D. Tildesley. *Computer Simulation of Liquids*. Clarendon Press, Oxford, UK (1987).

[16] P. L. DeVries. *A first course in computational physics*. John Wiley & Sons, New York (1993).

[17] W. Hoover. *Physical Review A*, **31** pp. 1695–1697 (1985).

[18] W. Hoover. *Physical Review A*, **34** pp. 2499–2500 (1986).

[19] W. Humphrey, A. Dalke, and K. Schulten. *Journal of Molecular Graphics*, **14** pp. 33–38 (1996).

[20] C. M. Neal, A. K. Starace, M. F. Jarrold, K. Joshi, S. Krishnamurty, and D. G. Kanhere. *Journal of Physical Chemistry C*, **111** pp. 17788–17794 (2007).

[21] S. Weiner, P. Kollman, D. Case, U. Singh, C. Ghio, G. Alagona, S. Profeta, and P. Weiner. *Journal of the American Chemical Society*, **106** pp. 765–784 (1984).

[22] S. Plimpton. *Journal of Computational Physics*, **117** pp. 1–19 (1995).

[23] J. C. Phillips, R. Braun, W. Wang, J. Gumbart, E. Tajkhorshid, E. Villa, C. Chipot, R. D. Skeel, L. Kalé, and K. Schulten. *J. Comp. Chem.*, **26** pp. 1781–1802 (2005).

[24] J. A. Anderson and A. Travesset. *Macromolecules*, **39** pp. 5143–5151 (2006).

[25] G. Whitesides and M. Boncheva. *Proceedings of the National Academy of Sciences of the United States of America*, **99** pp. 4769–4774 (2002).

[26] R. Kamien. *Science*, **299** pp. 1671–1673 (2003).

[27] M. Muthukumar, C. Ober, and E. Thomas. *Science*, **277** pp. 1225–1232 (1997).

[28] D. Bucknall and H. Anderson. *Science*, **302** pp. 1904–1905 (2003).

[29] F. Tanaka. *Polymer Journal*, **34** pp. 479–509 (2002).

[30] A. Semenov, J. Joanny, and A. Khokhlov. *Macromolecules*, **28** pp. 1066–1075 (1995).

[31] F. Tanaka. *Journal of Non-Crystalline Solids*, **307** pp. 688–697 (2002).

[32] B. Sung and A. Yethiraj. *Journal of Chemical Physics*, **119** pp. 6916–6924 (2003).

[33] P. Alexandridis and B. Lindman (eds.). *Amphiphilic Block Copolymers: Self-Assembly and Applications*. Elsevier Science, Amsterdam (2000).

[34] A. Kabanov, E. Batrakova, and V. Alakhov. *Journal of Controlled Release*, **82** pp. 189–212 (2002).

[35] M. Matsen. *Journal of Physics-Condensed Matter*, **14** pp. R21–R47 (2002).

[36] D. Pochan, Z. Chen, H. Cui, K. Hales, K. Qi, and K. Wooley. *Science*, **306** pp. 94–97 (2004).

[37] S. Jain and F. Bates. *Science*, **300** pp. 460–464 (2003).

[38] B. Cho, A. Jain, S. Gruner, and U. Wiesner. *Science*, **305** pp. 1598–1601 (2004).

[39] P. Ziherl and R. Kamien. *Journal of Physical Chemistry B*, **105** pp. 10147–10158 (2001).

[40] G. Grason, B. DiDonna, and R. Kamien. *Physical Review Letters*, **91** (2003).

[41] S. Asayama, A. Maruyama, C. Cho, and T. Akaike. *Bioconjugate Chemistry*, **8** pp. 833–838 (1997).

[42] Y. Chushak and A. Travesset. *Journal of Chemical Physics*, **123** (2005).

[43] P. Khalatur, A. Khokhlov, and D. Mologin. *Journal of Chemical Physics*, **109** pp. 9602–9613 (1998).

[44] D. Bedrov, G. Smith, and J. Douglas. *Europhysics Letters*, **59** pp. 384–390 (2002).

[45] L. Guo and E. Luijten. *Journal of Polymer Science Part B-Polymer Physics*, **43** pp. 959–969 (2005).

[46] G. Wanka, H. Hoffmann, and W. Ulbricht. *Macromolecules*, **27** pp. 4145–4159 (1994).

[47] B. Widom, P. Bhimalapuram, and K. Koga. *Physical Chemistry Chemical Physics*, **5** pp. 3085–3093 (2003).

[48] E. Dormidontova. *Macromolecules*, **35** pp. 987–1001 (2002).

[49] P. J. Flory. *Statistical Mechanics of Chain Molecules*. John Wiley & Sons, New York, NY (1969).

[50] J. Polson and N. Moore. *Journal of Chemical Physics*, **122** (2005).

[51] S. Mendez and J. Curro. *Macromolecules*, **37** pp. 1980–1986 (2004).

[52] *DL_POLY is a package of molecular simulation routines written by W. Smith and T.R. Forester, copyright The Council for the Central Laboratory of the Research Councils, Daresbury Laboratory at Daresbury, Nr. Warrington, UK, 1996.*

[53] M. Kroger. *Computer Physics Communications*, **118** pp. 278–298 (1999).

[54] P. Maiti, Y. Lansac, M. Glaser, N. Clark, and Y. Rouault. *Langmuir*, **18** pp. 1908–1918 (2002).

[55] G. Brassard and P. Bratley. *Fundamentals of Algorithmics*. Prentice Hall, New Jersey (1996).

[56] W. DeLano. 'The pymol molecular graphics system.' On the World Wide Web http://www.pymol.org (2002).

[57] A. Halperin and E. Zhulina. *Europhysics Letters*, **16** pp. 337–341 (1991).

[58] K. Mortensen, W. Brown, and E. Jorgensen. *Macromolecules*, **27** pp. 5654–5666 (1994).

[59] J. Shelley, M. Shelley, R. Reeder, S. Bandyopadhyay, and M. Klein. *Journal of Physical Chemistry B*, **105** pp. 4464–4470 (2001).

[60] H. Fukunaga, J. Takimoto, and M. Doi. *Journal of Chemical Physics*, **116** pp. 8183–8190 (2002).

[61] G. Srinivas, D. Discher, and M. Klein. *Nature Materials*, **3** pp. 638–644 (2004).

[62] A. Grosberg and A. Khokhlov. *Statistical Physics of Macromolecules*. AIP Press, Woodbury, New York (1994).

[63] A. Khokhlov, A. Semenov, and A. Subbotin. *European Physical Journal E*, **17** pp. 283–306 (2005).

[64] J. A. Anderson, C. D. Lorenz, and A. Travesset. *The Journal of Chemical Physics*, **128** p. 184906 (2008).

[65] C. N. Likos. *Soft Matter*, **2** pp. 478–498 (2006).

[66] B. Chu. *Langmuir*, **11** pp. 414–421 (1995).

[67] K. Mortensen. *Colloids and Surfaces A-Physicochemical and Engineering Aspects*, **183** pp. 277–292 (2001).

[68] G. McConnel, A. Gast, J. Huang, and S. Smith. *Physical Review Letters*, **71** pp. 2102–2105 (1993).

[69] J. Bang, T. Lodge, X. Wang, K. Brinker, and W. Burghardt. *Physical Review Letters*, **89** (2002).

[70] T. Lodge, B. Pudil, and K. Hanley. *Macromolecules*, **35** pp. 4707–4717 (2002).

[71] T. Lodge, J. Bang, M. Park, and K. Char. *Physical Review Letters*, **92** p. 145501 (2004).

[72] J. Bang and T. Lodge. *Physical Review Letters*, **93** p. 245701 (2004).

[73] G. McConnell and A. Gast. *Physical Review E*, **54** pp. 5447–5455 (1996).

[74] B. van Vlimmeren, N. Maurits, A. Zvelindovsky, G. Sevink, and J. Fraaije. *Macromolecules*, **32** pp. 646–656 (1999).

[75] Y. Lam and G. Goldbeck-Wood. *Polymer*, **44** pp. 3593–3605 (2003).

[76] X. Zhang, S. Yuan, and J. Wu. *Macromolecules*, **39** p. 6631 (2006).

[77] G. M. Grason. *Journal of Chemical Physics*, **126** (2007).

[78] D. Bedrov, C. Ayyagari, and G. Smith. *Journal of Chemical Theory and Computation*, **2** pp. 598–606 (2006).

[79] K. Schmidt-Rohr. *Journal of Applied Crystallography*, **40** pp. 16–25 (2007).

[80] B. M. Mladek, P. Charbonneau, and D. Frenkel. *Physical Review Letters*, **99** (2007).

[81] D. Frenkel and B. Smit. *Understanding Molecular Simulations*. Academic Press, San Diega, CA (2002).

[82] M. Doi and S. Edwards. *The Theory of Polymer Dynamics*. Clarendon Press, Oxford, UK (2001).

[83] P. Steinhardt, D. Nelson, and M. Ronchetti. *Physical Review B*, **28** pp. 784–805 (1983).

[84] C. Kittel. *Introduction to Solid State Physics*. John Wiley & Sons, Hoboken, NJ (2005).

[85] J. Hansen and I. McDonald. *Theory of Simple Liquids*. Academic Press, London, UK (1986).

[86] A. Semenov. *Sov. Phys. JETP*, **61** p. 733 (1985).

[87] M. Watzlawek, C. Likos, and H. Lowen. *Physical Review Letters*, **82** pp. 5289–5292 (1999).

[88] S. Alexander and J. McTague. *Physical Review Letters*, **41** pp. 702–705 (1978).

[89] B. Groh and B. Mulder. *Physical Review E*, **59** pp. 5613–5620 (1999).

[90] W. Klein. *Physical Review E*, **64** (2001).

[91] W. Ostwald. *Z. Phys. Chem (Leipzig)*, **22** p. 289 (1897).

[92] L. Leibler. *Macromolecules*, **13** pp. 1602–1617 (1980).

[93] C. Marques and M. Cates. *Europhysics Letters*, **13** pp. 267–272 (1990).

[94] R. Prudhomme, G. Wu, and D. Schneider. *Langmuir*, **12** pp. 4651–4659 (1996).

[95] K. Mortensen and Y. Talmon. *Macromolecules*, **28** pp. 8829–8834 (1995).

[96] T. Liu and B. Chu. *Journal of Applied Crystallography*, **33** p. 727 (2000).

[97] Y. L. et al. *J. Phys. Chem. B.*, **110** p. 26424 (2006).

[98] D. C. Pozzo and L. M. Walker. *Colloids and Surfaces A-Physicochemical and Engineering Aspects*, **294** pp. 117–129 (2007).

[99] D. Enlow, A. Rawal, M. Kanapathipillai, K. Schmidt-Rohr, S. Mallapragada, C. T. Lo, P. Thiyagarajan, and M. Akinc. *Journal of Materials Chemistry*, **17** pp. 1570–1578 (2007).

[100] 'Cuda programming guide 1.1.' http://developer.nvidia.com/object/cuda.html.

[101] H. J. C. Berendsen, D. van der Spoel, and R. van Drunen. *Comp. Phys. Comm.*, **91** pp. 43–56 (1995).

[102] H.-J. Limbach, A. Arnold, B. A. Mann, and C. Holm. *Comput. Phys. Commun*, **174** pp. 704–727 (2006).

[103] S. Kupka. In 'CESCG,' (2006).

[104] J. Yang, Y. Wang, and Y. Chen. *J. Comp. Phys.*, **221** pp. 799–804 (2007).

[105] Z. Yao, J. Wang, G. Liu, and M. Cheng. *Comp. Phys. Comm.*, **161** pp. 27–35 (2004).

[106] T. Maximova and C. Keasar. *J. Comp. Biol.*, **13** pp. 1041–1048 (2006).

[107] H. Han and C.-W. Tseng. In '5th Workshop on Languages, Compilers, and Run-time Systems for Scalable Computers (LCR'2000),' pp. 253–275 (2000).

[108] B. Moon, H. Jagadish, C. Faloutsos, and J. H. Saltz. *IEEE Transactions on Knowledge and Data Engineering*, **13** (2001).

[109] J. Wang and J. Shan. In 'Geocomputation,' (2005).

[110] S. D. Bond, B. J. Leimkuhler, and B. B. Laird. *J. Comp. Phys.*, **151** pp. 114–134 (1999).

[111] M. Tuckerman, B. J. Berne, and G. J. Martyna. *The Journal of Chemical Physics*, **97** pp. 1990–2001 (1992).

[112] Y. Ng., C. Chan, R. R. Schrock, and R. E. Cohen. *Macromolecules*, **4** p. 24 (2007).

[113] M. Park, C. Harrison, P. M. Chaikin, R. A. Register, and D. H. Adamson. *Science*, **276** p. 1401 (1997).

[114] M. R. Bockstaller, Y. Lapetnikov, S.Margel, and E. L. Thomas. *JACS*, **125** p. 5276 (2003).

[115] J. J. Chiu, B. J. Kim, E. J. Kramer, and D. J. Pine. *JACS*, **127** p. 5036 (2005).

[116] B. J. Kim, J. Bang, C. J. Hawker, and E. J. Kramer. *Macromolecules*, **39** p. 4108 (2005).

[117] A. Balazs, T. Emrick, and T. P. Russell. *Science*, **314** p. 1107 (2006).

[118] B. J. Kim, G. H. Fredrickson, C. J. Hawker, and E. J. Kramer. *Langmuir*, **23** p. 7804 (2007).

[119] S. C. Park, B. J. Kim, C. J. Hawker, E. J. Kramer, J. Bang, and J. S. Ha. *Macromolecules*, **40** p. 8119 (2007).

[120] R. D. Deshmukh, G. A. Buxton, N. Clarke, and R. J. Composto. *Macromolecules*, **40** p. 6316 (2007).

[121] C. T. L. et al. *Macromolecules*, **40** p. 8302 (2007).

[122] R. B. Thompson, V. V. Ginzburg, M. W. Matsen, and A. C. Balazs. *Science*, **292** p. 2469 (2001).

[123] R. B. Thompson, V. V. Ginzburg, M. W. Matsen, and A. C. Balazs. *Macromolecules*, **35** p. 1060 (2002).

[124] D. Bedrov, G. D. Smith, and J. S. Smith. *J. Chem. Phys.*, **119** p. 10438 (2003).

[125] A. J. Schultz, C. K. Hall, and J. Genzer. *Macromolecules*, **38** p. 3007 (2005).

[126] S. W. Sides, B. J. Kim, E. J. Kramer, and G. H. Fredrickson. *Phys. Rev. Lett*, **96** p. 250601 (2006).

[127] V. Pryamitsyn and V. Ganesan. *Macromolecules*, **39** p. 8499 (2006).

[128] F. J. Martinez-Veracoechea and F. A. Escobedo. *Macromolecules*, **40** p. 7354 (2008).

[129] V. Ganesan. *J. of Polymer Science B*, **46** p. 2666 (2008).

[130] M. Kanapathipillai, Y. Yusufoglu, A. Rawal, Y.-Y. Hu, C. T. Lo, P. Thiyagarajan, Y. E. Kalay, M. Akinc, S. Mallapragada, and K. Schmidt-Rohr. *Chem. Mater*, **20** p. 5922 (2008).

[131] J. E. Magee and F. R. Siperstein. *J. Phys. Chem. C*, **113** p. 1680 (2009).

[132] S. C. Glotzer and M. J. Solomon. *Nature Materials*, **6** p. 557 (2007).

[133] S. Milner, T. Witten, and M. Cates. *Macromolecules*, **21** pp. 2610–2619 (1988).

[134] M. Daoud and J. Cotton. *Journal De Physique*, **43** pp. 531–538 (1982).

[135] E. W. Cochran, C. J. Garcia-Cervera, and G. H. Fredrickson. *Macromolecules*, **39** pp. 2449–2451 (2006).

[136] P. Alexandridis, U. Olsson, and B. Lindman. *Langmuir*, **14** pp. 2627–2638 (1998).

[137] E. Eiser, F. Molino, G. Forte, and X. Pithon. *Rheologica Acta*, **39** pp. 201–208 (2000).

[138] J. Stone. *An efficient library for parallel ray tracing and animation*. Master's thesis, Computer Science Department, University of Missouri-Rolla (1998).

[139] C. R. Iacovella, A. S. Keys, M. A. Horsch, and S. C. Glotzer. *Physical Review E*, **75** (2007).

[140] L. Walker. private communication (2009).

[141] D. Goldberg. *ACM Computing Surveys*, **32** p. 548 (1991).

[142] R. Hockney and J. Eastwood. *Computer simulation using particles*. Institute of Physics Publishing, London (1988).

[143] D. Göddeke, R. Strzodka, and S. Turek. *International Journal of Parallel, Emergent and Distributed Systems*, **22** pp. 221–256 (2007).

[144] G. Brannigan, P. F. Philips, and F. L. H. Brown. *Phys. Rev. E*, **72** p. 011915 (2005).

[145] Y. Li, T. Shi, Z. Sun, L. An, and Q. Huang. *Journal of Physical Chemistry B*, **110** pp. 26424–26429 (2006).