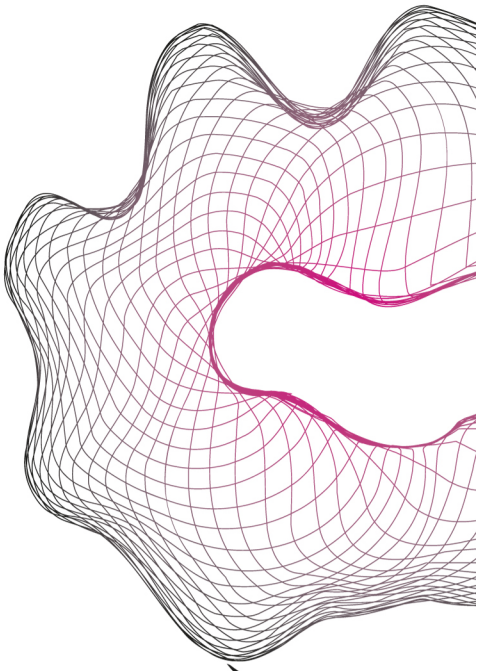

COARSE-GRAINED MOLECULAR DYNAMICS SIMULATIONS TO STUDY VAPOUR SOLVATION IN POLYMER BRUSHES

ON THE BORDER BETWEEN POLYMER PHYSICS AND COMPUTATIONAL SCIENCE

Lars Veldscholte

Materials science and Technology of Polymers,
Faculty of Science and Technology /
MESA+ Institute for Nanotechnology,
University of Twente

Master's thesis
M.Sc. in Chemical Engineering,
Molecular & Materials Engineering

Committee:
Prof. dr. G.J. Vancso
Dr. S.J.A. de Beer
Dr. ir. W.K. den Otter

UNIVERSITY OF TWENTE.

JANUARY 2020

Abstract

Polymer brushes, films consisting of polymers densely grafted to a surface, are of interest for a range of applications due to their intriguing behaviours in solvents. Although their behaviour in liquid solvents has been studied extensively, their behaviour in solvent vapours has not.

In this work, Molecular Dynamics (MD) simulations of two different coarse-grained models of polymer brushes are employed to investigate the sorption behaviour of polymer brushes in solvent vapours. A constant pressure solvent environment was simulated using a grand-canonical Monte-Carlo (GCMC) chemostat. For polymers modelled by the Kremer-Grest (KG) model, a 2D parameter sweep of polymer self-affinity and polymer-solvent affinity was performed, after which sorption behaviour is classified by analysing density profiles.

The KG brush exhibits strong swelling induced by absorption that appears to be governed by the relative affinity of polymer-solvent, to polymer with itself. Adsorption on the other hand, which presents as a layer of solvent on top of the brush, appears to be governed by polymer-solvent affinity only. A simulation of polyethylene (PE) in acetone, both modelled by the MARTINI model revealed density profiles similar to the results obtained from a KG brush with low-to-moderate polymer-solvent and polymer self-affinities.

The collapse of the absorption of solvent in the brush onto a single parameter (the relative affinity) suggests that this aspect of the system's behaviour is not influenced by chain stretching entropy. The MARTINI simulation of PE in acetone confirms the KG simulations and illustrates the potential for simulating chemically different polymers and solvents.

Contents

Preface

Dear reader,

In front of you is a copy of my master's thesis.

I got involved with this project about the simulation of vapour solvation of polymer brushes after a visit to the Forschungszentrum Jülich with Sissi de Beer, Karel van der Weg, and Guido Ritsema van Eck to consult some HPC experts at the Jülich Supercomputing Centre about the scaling optimisation of our simulations. During that visit, we first thought about the idea to automate the sampling of a 2D phase diagram by MD simulations. It led to me writing some Python scripts as wrappers around LAMMPS that automatically classified the sorption regime a system was in, and sampling the next point in the 2D parameter space based on that. The scripts went through a few iterations as our physical understanding of the system improved and I got more involved with the project.

It was interesting, yet somewhat unexpected, to spend a full master's assignment doing theoretical and computational work. At the start of my assignment, I set on continuing (experimental) research on ultra-thick enzymatically-grown biopolymer brushes consisting of hyaluronan that I worked on during my internship in a collaboration with Jennifer Curtis and Jessica Faubel from the Curtis group at Georgia Tech in Atlanta. However, for logistic reasons, we decided to drop the hyaluronan project and focus on the simulation work. Of course, this means that any possible overlap between Guido's thesis and mine is not coincidental, as we collaborated closely on the same subject. In our joint effort, Guido's focus was mostly the theory while I was responsible for most of the simulations and computational aspects.

In the first chapter of this thesis, I introduce you to the problem at hand (polymer brushes and their behaviour in solvent vapour) and the main research method (MD simulations). After a short primer on polymer physics, I go in-depth about the theory of molecular dynamics before reporting the conducted simulations and discussing their results. The appendices deal with topics that are not directly related to polymer physics: Appendix A contains reports on various pieces of software I wrote during this period to aid the generation, analysis, and automation of my simulation work, with some of the more polished pieces of code also available on my GitHub. I also had the opportunity to design and build a new (32-core!) AMD Threadripper machine for high-performance MD simulation, intended to replace the four old PCs we had in use for that purpose before. This build was finished just before the conclusion of my master's assignment, and is described in detail in Appendix B in the broader context of LAMMPS performance and scaling.

Before I forget, I'd like to thank my graduation committee, consisting of Julius Vancso, Sissi de Beer, and Wouter den Otter for their time and commitment, as well as Jos Paulusse who withdrew from the committee but whose input during the midterm was much appreciated. I especially want to express my gratitude to my supervisor Sissi for her everlasting optimism and support, and for all the times she managed to help and motivate me. I don't think I could wish for a better mentor. Of course Guido, with whom I worked intimately on this project, was invaluable for the in-depth discussions we had on the subject. I want to acknowledge SURFSara for HPC resources; Having access to Cartesius is a bliss compared to running simulations on our own PCs. I probably also wouldn't have figured out a lot of things if it were not for the people on the `lammps-users` mailing list who responded to my questions when I got stuck, among them several LAMMPS lead developers. I want to acknowledge Jan Meinke, Ilya Zhukov, Sandipan Mohanty and Olav Zimmermann from the Jülich Supercomputing Centre for their advice on performance optimisation of the simulations. Finally, my fellow MTP students deserve a honourable mention for the good company in the office.

Lars Veldscholte

List of Symbols

N	Number of particles, polymer chain length
R_g	Radius of gyration
ρ_g	Grafting density
n	Moles of particles, generic integer
V	Volume
E	Energy
T	Temperature
P	Pressure
μ	Chemical potential
t	Time
Δt	Timestep
r	Spatial coordinate, radial distance
v	Velocity
a	Acceleration
F	Force
U	Potential
ϵ	Lennard-Jones energy
σ	Lennard-Jones zero-crossing distance
r_c	Potential cut-off distance
r_m	Lennard-Jones potential minimum
d_f	Degrees of freedom
k	Boltzmann constant
m	Mass
q	Charge
ρ	(Number) density
A	Area
τ	Time constant
β	Thermodynamic beta

List of Figures

List of Tables

1 | Introduction

1.1 Polymer brushes

A polymer brush is a film consisting of polymer chains grafted with one end to the surface, at such a density that their ideal volumes overlap and they extend away from the surface and form a 'brush' structure (Figure 1.1), named by analogy with the arrangement of hairs on a brush.

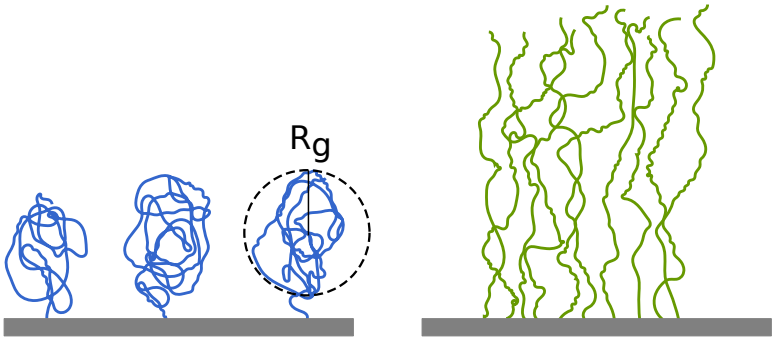

Figure 1.1: *Polymers grafted to a surface with a density below the critical grafting density form mushrooms (**left**). If the grafting density is sufficiently higher than the critical grafting density, a brush is formed (**right**).*

Polymer brushes have great potential as a means to control surface properties [1, 2]. As such, surface functionalisation by polymer brushes has applications in stabilisation of colloids [1], stimulus-responsive systems (sensors) [3, 4], anti-fouling surfaces [5], low-friction surfaces [6, 7], smart adhesives [8], among others.

1.1.1 Vapour hydration

For many of these applications, the polymer brushes are assumed to be immersed in a liquid solvent. Yet, in many circumstances it is convenient if these functionalised surfaces can also be applied in air. The primary constituents of air (nitrogen and oxygen) are obviously always poor solvents, but air is seldom dry and fortunately water is a good solvent for many polymers. Examples of applications of polymer brushes where this is relevant are vapour-hydrated lubricants and organic vapour sensors.

Thus, if we want to apply polymer brushes in humid air or other solvent vapours, it is imperative to understand how polymer brushes are solvated by vapours. Yet, most of the knowledge about polymer brush solvation is about liquid solvation and significantly less is known about the vapour-solvation of polymer brushes. [9]

In most theoretical studies, Flory-Huggins theory is employed to model the solvation of polymer brushes. However, this model fails to capture interfacial effects as it assumes a homogeneous distribution of solvent throughout the brush. Yet, neutron reflectometry experiments indicate that this distribution is not homogeneous, and that in some cases an enhancement of solvent density at the interface of the brush exists. [10, 11]

1.2 Research method

In this work, Molecular Dynamics (MD) is utilised to study polymer brush solvation. MD simulations have the advantage that they are easier and less expensive to set-up than experiments in the laboratory; interactions between components can be freely defined and all particles' properties are directly observable, unlike in real systems where e.g. density profiles can be tricky to measure.

In order to be able to simulate the behaviour of brushes in a constant concentration/-pressure environment, a *chemostat* is required. MD in its most simple form results in closed systems, where neither energy nor particles can be exchanged with the environment. Thermostats (3.4.1) are an established mechanism in MD to achieve a constant temperature by exchanging energy with the environment. However, achieving a constant *chemical potential* is more convoluted, as this is not possible with strictly MD.

In his master's assignment [12], Jan-Willem Nijkamp laid the basis by applying the GCMC chemostat, a hybrid MD/MC (Monte-Carlo) mechanism for exchanging particles with the environment with the goal of achieving a constant chemical potential (3.4.2), to simulations of vapour hydration of polymer brushes. This work builds on Jan-Willem's earlier work in this field, as I improved the computational performance of the simulation by optimising balancing of subdomains (B.2.1), thoroughly investigated thermodynamic properties of the simulated solvent (4.1.1), more comprehensively probed the polymer self-affinity and polymer-solvent affinity parameter space to reveal sorption regimes (4.2.2), and extended the scope of these simulations to a more chemically-specific coarse-graining model (4.1.3). Moreover, I developed wrapper scripts around LAMMPS in Python that streamline simulating large batches of these specific simulations, a more flexible and extensible generator for initial data (A.2) and scripts that aid the analysis of simulation results.

2 | Theory of polymer brushes

2.1 Free polymers in solution

Polymers in solution take on different conformations depending on their interactions with the solvent. The size of a polymer coil, often measured by its radius of gyration R_g, is described by the balance of two forces: the entropic contraction, driven by the random motion (self-avoiding walk) of the chain, and the enthalpic extension that depends on the interaction with the solvent. Depending on how the radius of gyration scales with the chain length N, three solvent regimes can be distinguished: [13]

- Good solvent conditions: $R_g \propto N^{3/5}$
- Theta conditions: $R_g \propto N^{1/2}$
- Poor solvent conditions: $R_g \propto N^{1/3}$

If the polymer is solvated by a good solvent, the coil will expand to maximise the interactions with the solvent. If the solvent is poor, the coil will contract, forming a dense globule. A special case exists when the solvent interactions precisely cancel out excluded volume effects in the chain, resulting in the chain assuming an ideal random walk conformation, like in a melt. The latter is called the *theta condition*. [13]

As the entropic contribution to the chain energy depends on temperature, and the enthalpic ones depend on the specific polymer and solvent, above conditions are only meaningfully defined for a certain set of polymer and solvent at a given temperature.

2.2 Polymer brushes

When polymers are grafted onto a surface, they lose considerable entropy since one end of their chains is fixed in space and the space behind the grafting surface is inaccessible, but their conformation is still influenced by the solvent. When the chains are grafted close enough that their ideal volumes overlap, they will stretch away from the surface and form a structure called a polymer brush (Figure 1.1). This happens at the critical grafting density: the point that the (mean) distance between grafting points is closer than two times the (mean) radius of gyration of a free polymer: [13]

$$\frac{1}{\sqrt{\rho_g}} = 2R_g \tag{2.1}$$

$$\rho_g{}^* = \frac{1}{4R_g^2} \tag{2.2}$$

where ρ_g denotes the grafting density, $\rho_g{}^*$ the critical grafting density, and R_g the radius of gyration. From this derivation, it also follows that the critical grafting density is dependent on solvent quality.

In this case the polymer brush height is dependent on both the grafting density and the length (Equation 2.3). The swelling response when solvated by good solvents is enhanced, because the height increases linearly with chain length since chains can no longer expand in the two directions parallel to the grafting surface. The scaling with grafting density is characterised by a parameter ν, which depends non-trivially on the solvent quality.

$$h \propto \rho_g^{\nu} N \tag{2.3}$$

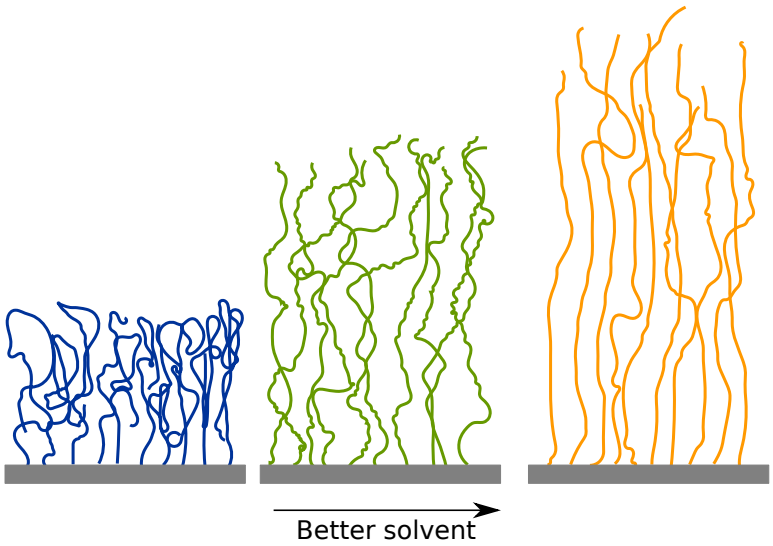

Figure 2.1: *Polymer brushes collapse in poor solvents and swell in good solvents by the stretching behaviour of the individual chains.*

3 | Theory of molecular dynamics

MD is a simulation method for studying physical behaviour of systems on a molecular scale. The basic principle of MD is that trajectories of particles (atoms/molecules) are simulated by numerically integrating Newton's equations of motion, while they interact with each other by pair potentials (also called 'force fields'). A pair potential gives the potential (energy) of a pair of interacting particles as function of the distance between them. Differentiating this potential gives the force acting on those particles. Every 'tick' of the MD simulation, the positions and speeds of all particles are updated according to the forces acting on them.

By evolving a system in time, a statistical ensemble can be sampled depending on how the simulation is set-up. For example, integrating without thermostatting yields a microcanonical ensemble (NVE). Thermostats or barostats can be utilised to generate canonical (NVT) or isothermal-isobaric (NPT) ensembles respectively.

MD can be used to study a wide range of systems and phenomena in physics, material science, and chemistry. Applications include but are not limited to dynamics of solid state physics that are hard to observe in real-life, resolving structures of macromolecules such as proteins and polymers, and supramolecular interactions.

3.1 Integration schemes

Simply said, the main principle behind MD is to numerically integrate the equations of motion of an ensemble of many particles. This is realised by discretising time by finite difference methods. [14]

All numeric integration inevitably leads to some error, manifesting as total energy drift in MD. An important point to realise is that for MD, we are not so much concerned with energy fluctuations on a short time scale, as long as the energy is conserved on long time scales. One may be inclined to think that it is important for the algorithm to accurately simulate the trajectories of all particles on all time scales, but that turns out not to be the case. Many-body systems are generally chaotic: that is, the exact trajectories are very sensitive to initial conditions, and two initially close trajectories are always expected to diverge exponentially as time evolves. The inevitable integration error has the same effect on the trajectories, no matter how small small it is. [15]

This is not a problem because we are not interested in the detailed, accurate simulation of the trajectory of a chaotic system with carefully chosen initial conditions, like would be the case for a simulation of orbital mechanics, for example: in that case we want to know the exact trajectory of the system and any divergence in that trajectory is a big issue. In the case of MD, we are only interested in the ensemble average behaviour of a system in a given macrostate (e.g. a system with a set total energy). It does not matter if the exact trajectories diverge, as long as the resulting trajectory represents *a* valid trajectory and statistical predictions, such as thermodynamic properties like temperature and pressure, can still be made. [15]

Another important criterium is time reversibility. This means that the integration scheme should be symmetric with respect to time, e.g. if we reverse the momenta of all particles in the system, the system should exactly trace back its trajectory. After all, Newton's equations of motion are time-reversible, so our integration scheme should be, too. The energy-preserving and time-reversible requirements are fulfilled by so-called symplectic integrators. [14, 15]

3.1.1 Verlet

A simple symplectic scheme that satisfies the criteria discussed above is the Verlet algorithm.

As usual with finite difference schemes, the derivation starts with the Taylor expansion of a particle coordinate (r) around time t. The next two expressions give the approximate coordinates one timestep (Δt) after and before t:

$$r(t+\Delta t) = r(t) + \frac{\mathrm{d}r(t)}{\mathrm{d}t}\Delta t + \frac{\mathrm{d}^2 r(t)}{\mathrm{d}t^2}\frac{\Delta t^2}{2} + \frac{\mathrm{d}^3 r(t)}{\mathrm{d}t^3}\frac{\Delta t^3}{3!} + O(\Delta t^4) \tag{3.1}$$

$$r(t-\Delta t) = r(t) - \frac{\mathrm{d}r(t)}{\mathrm{d}t}\Delta t + \frac{\mathrm{d}^2 r(t)}{\mathrm{d}t^2}\frac{\Delta t^2}{2} - \frac{\mathrm{d}^3 r(t)}{\mathrm{d}t^3}\frac{\Delta t^3}{3!} + O(\Delta t^4). \tag{3.2}$$

The Verlet scheme can be obtained by considering the central difference approach to the second derivative, which is obtained by the sum of the two expressions above:

$$r(t+\Delta t) + r(t-\Delta t) = 2r(t) + \frac{\mathrm{d}^2 r(t)}{\mathrm{d}t^2}\Delta t^2 + O(\Delta t^4). \tag{3.3}$$

Rearranging gives:

$$\begin{aligned} r(t+\Delta t) &= 2r(t) - r(t-\Delta t) + \frac{\mathrm{d}^2 r(t)}{\mathrm{d}t^2}\Delta t^2 + O(\Delta t^4) \\ r(t+\Delta t) &= 2r(t) - r(t-\Delta t) + a(t) \quad \Delta t^2 + O(\Delta t^4) \end{aligned} \tag{3.4}$$

where $\frac{\mathrm{d}^2 r(t)}{\mathrm{d}t^2} = a(t)$ is the acceleration, which is given by the total force acting on a particle $F(t)$ divided by its mass m. The truncation error is of order Δt^4, because the conveniently cancelling odd terms in the Taylor expansion. This also makes the scheme time reversible.

Note that in the Verlet scheme, the velocities are not explicitly given. This is problematic for MD because the velocities are required for the computation of kinetic energy and other thermodynamic properties derived from that. They can be computed using the central difference approach to the first derivative, which is obtained by subtracting Equation 3.2 from 3.1:

$$r(t+\Delta t) - r(t-\Delta t) = 2\frac{\mathrm{d}r(t)}{\mathrm{d}t}\Delta t + O(\Delta t^3). \tag{3.5}$$

Recognising that $\frac{\mathrm{d}r(t)}{\mathrm{d}t} = v(t)$ and rearranging gives:

$$v(t) = \frac{r(t+\Delta t) - r(t-\Delta t)}{2\Delta t} + O(\Delta t^2). \tag{3.6}$$

The above expression for the velocity is accurate to second order in Δt. The Verlet scheme is also not self-starting, because a value for $r(t-\Delta t)$ is required which has to be provided by another scheme.

3.1.2 Velocity Verlet

A scheme that is functionally equivalent to Verlet, but explicitly computes velocities is the so-called velocity-Verlet scheme.

Here, the particle coordinates are computed by a second-order forward difference:

$$r(t+\Delta t) = r(t) + \frac{\mathrm{d}r(t)}{\mathrm{d}t}\Delta t + \frac{\mathrm{d}^2 r(t)}{\mathrm{d}t^2}\frac{\Delta t^2}{2} + O(\Delta t^3). \tag{3.7}$$

Velocities are computed separately:

$$\begin{aligned} v(t+\Delta t) &= v(t) + \frac{\mathrm{d}v(t)}{\mathrm{d}t}\Delta t + \frac{\mathrm{d}^2 v(t)}{\mathrm{d}t^2}\frac{\Delta t^2}{2} + O(\Delta t^3) \\ v(t+\Delta t) &= v(t) + a(t)\Delta t + \frac{\mathrm{d}^2 v(t)}{\mathrm{d}t^2}\frac{\Delta t^2}{2} + O(\Delta t^3). \end{aligned} \tag{3.8}$$

Unlike the Verlet scheme, velocity-Verlet is self-starting and velocities are defined explicitly. Additionally, it can be shown that the scheme is fully equivalent to Verlet; that is, its accuracy is identical and for the same systems and initial conditions, it generates identical trajectories.

3.1.3 rRESPA

rRESPA (reversible REference System Propagator Algorithm) [16] is a multi time scale integrator. It functions similarly to velocity-Verlet, but features an arbitrary number of hierarchical iteration levels which allow some interactions to be computed more frequently than others.

Generally, it is desirable to make the timestep Δt as long as possible for efficiency reasons. However, when the timestep is too long, it can result (depending on the system) in numerical instabilities, manifesting as energy drift or even 'exploding' systems in MD. As a rule of thumb, the timestep should be roughly comparable to the resonance timescales of the highest frequency modes in the system in order for the simulation to be stable. For example, in an all-atom simulation, this would be the bond vibrations of hydrogen atoms which require a timestep of around $0.5\,\mathrm{fs}$. [17]

rRESPA can make the simulation more efficient by computing some interactions more frequently. It defines an outer timestep and a number of hierarchical inner loops. For example, it is possible to setup a 2-level rRESPA integration where the inner loop is executed 10 times per outer timestep. Hence it is possible to evaluate the high-frequency interactions such as the bonds to hydrogen atoms every $0.5\,\mathrm{fs}$, but evaluate the much more expensive non-bonded pair interactions only every $5\,\mathrm{fs}$.

Similarly, for coarse-grained models this can be of use because here too there is a separation of modes in terms of their time scales: bond, angle, and dihedral interactions typically require much shorter timesteps than non-bonded pair interactions.

Thus, with rRESPA it is possible to attain a speed-up over simple Verlet integration while retaining the same numerical stability.

3.2 Pair potentials

A pair potential in MD specifies the potential energy of two interacting particles as a function of their separation distance ($U(r)$). By differentiating with respect to r, the interparticle force is obtained:

$$F_{i,j} = -\frac{\mathrm{d}U(r_{i,j})}{\mathrm{d}r_{i,j}} \tag{3.9}$$

One can view the choice of a potential in MD as type of material one is simulating, as (in the absence of molecular topological features like bonds) the potential determines the majority of thermodynamic and mechanical properties.

3.2.1 Lennard-Jones

The Lennard-Jones potential (Equation 3.10, LJ from here on) is a commonly used potential for describing generic matter. It consists of an attractive term which represents the effective van der Waals interaction of atoms without a permanent electrostatic dipole moment (London dispersion), of which the potential scales with r^{-6}, and a repulsive term that scales with r^{-12} that is loosely modelled after Pauli exclusion, but has no rigorous theoretical justification. The r^{-12} term is mostly chosen for computational convenience, as it is the square of r^{-6}. [18]

$$U_{\mathrm{LJ}}(r) = 4\epsilon\left(\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right) \tag{3.10}$$

In Equation 3.10, ϵ represents the depth of the potential well (with dimensions of energy) and σ represents the zero-crossing distance. The minimum occurs at $r_m = 2^{1/6}\sigma$.

Because the LJ potential only takes the effects of London dispersion into account, it is in the first place strictly a model of monoatomic gasses. However, it enjoys a lot of popularity in MD as a model of 'generic fluid'. It is used combined with a bonded potential as the Kremer-Grest model of coarse-grained polymers (see 3.5.1) [19].

Truncation and shifting

In practical MD simulations, the LJ potential is usually truncated at a certain point called the cut-off distance (r_c) to make simulations of systems containing many particles interacting by the LJ potential computationally feasible. This is justifiable because the interaction is negligible at distances beyond the cut-off. Still, simply truncating the potential as in Equation 3.11 creates a jump discontinuity in the potential which leads to a force singularity at that point. To solve this, the potential is shifted up by the value of the potential at the cut-off distance $U_{\mathrm{LJ}}(r_c)$ (see Equation 3.12). This Shifted-Potential (SP) potential avoids the discontinuity in the potential, but still suffers from a discontinuity in the first derivative (the force). The Shifted-Force (SF) potential improves upon this by shifting the force so that it goes to zero continuously (see Figure 3.1). This is equivalent to subtracting a linear term from the potential (Equation 3.13). [20]

$$U_{\mathrm{LJ,cut}}(r) = \begin{cases} U_{\mathrm{LJ}}(r) & \text{for } r \leq r_c \\ 0 & \text{for } r > r_c \end{cases} \tag{3.11}$$

$$U_{\mathrm{LJ,SP}}(r) = \begin{cases} U_{\mathrm{LJ}}(r) - U_{\mathrm{LJ}}(r_c) & \text{for } r \leq r_c \\ 0 & \text{for } r > r_c \end{cases} \tag{3.12}$$

$$U_{\mathrm{LJ,SF}}(r) = \begin{cases} U_{\mathrm{LJ}}(r) - (r - r_c)U'_{\mathrm{LJ}}(r_c) - U_{\mathrm{LJ}}(r_c) & \text{for } r \leq r_c \\ 0 & \text{for } r > r_c \end{cases} \tag{3.13}$$

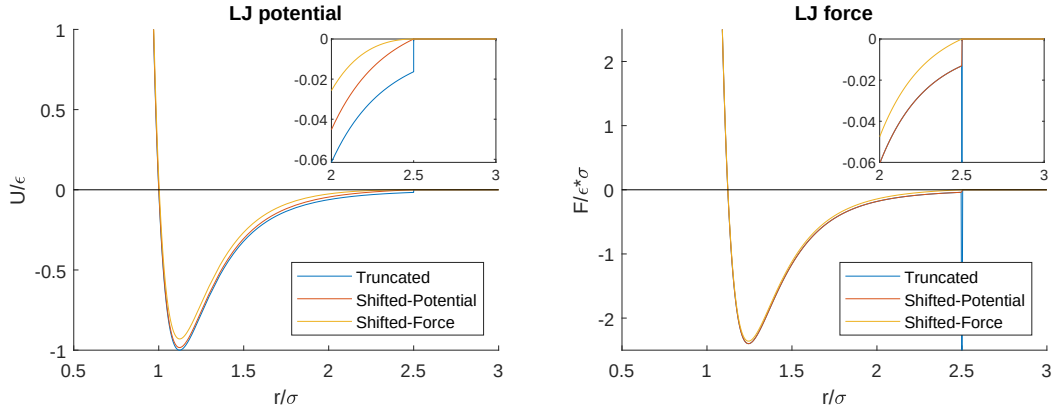

Figure 3.1: *The truncated LJ-potential ($r_c = 2.5\sigma$) has a jump discontinuity causing a singularity in the force. The SP potential is continuous, but its force is not. The SF potential is continuous in both the potential and the force.*

Note that in practical simulations, truncation means that all interactions between particles separated more than r_c are simply ignored. This means that the singularity in the force corresponding to the unshifted truncated potential is not encountered and the SP potential results in exactly the same *dynamics* as the unshifted truncated potential. The only tangible difference lies in the computed energies of the system; those will be incorrect for the unshifted case. What does make a difference for dynamics is the discontinuity in the force, as this leads to energy drift over time. The SF potential does not suffer from the energy drift while the SP potential does. [20]

The LJ potential is commonly truncated at $r_c = 2.5\,\sigma$. When the LJ potential is truncated at the energy minimum (occurring at $r_c = 2^{1/6}\,\sigma \approx 1.1225\,\sigma$), a purely repulsive potential known as the Weeks–Chandler–Andersen (WCA) potential is obtained. This is commonly utilised together with Langevin dynamics (see 3.4.1) to produce an implicit solvent: a continuum technique to mimic the net effect of a solvent without actually simulating the dynamics of the solvent particles. For the WCA potential it suffices to shift the potential, as the force is naturally zero at the cut-off point.

LJ-spline

The truncation problem of the LJ potential has spawned a number of additional, more sophisticated variants. One such variant is the Lennard-Jones Spline (LJ-spline). In this model, The Lennard-Jones potential is replaced with a cubic spline between the inflection point (r_s) and the cut-off (r_c). In Equation 3.14, r_c and A_3 are chosen in such a way that the cubic section is continuous at r_s and r_c. Like the truncated LJ potential, the thermodynamic properties are different from the full LJ potential. The advantage of the LJ-spline potential is an even lower computational cost compared to the LJ potential truncated at $2.5\,\sigma$. [21]

$$U_{\mathrm{LJ,spline}}(r) = \begin{cases} U_{\mathrm{LJ}}(r) & \text{for } r \le r_s \\ U_{\mathrm{LJ}}(r_s) - (r - r_s)U'_{\mathrm{LJ}}(r_s) - \frac{1}{6}A_3(r - r_s)^3 & \text{for } r_s < r \le r_c \\ 0 & \text{for } r > r_c \end{cases} \tag{3.14}$$

3.3 Neighbour lists

A big (and computationally expensive) part of MD is the computation of forces between all pairs of particles. If we do not apply any clever tricks, this essentially yields a n-body problem, which is notoriously hard and expensive to solve. [15, 22]

Pair potentials are, as discussed above, usually truncated beyond a certain distance. This means that at least the force calculation can be skipped for particles further away than the cut-off, but we still need to calculate the distance to every particle. As the number of pairs is equal to n choose 2:

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{(n-1)n}{2} \tag{3.15}$$

this implies that the algorithm has a time complexity of $O(n^2)$, i.e. the time needed scales quadratically with the total number of particles in the system. Neighbour lists (also called Verlet lists) can be used to improve this. [15, 22]

The principle behind neighbour lists is to consider all neighbouring particles around a particle within a radius r_n that is some r_s (the skin distance) larger than r_c (the cut-off radius) of the pair potential (see Figure 3.2) and put them in a list. This 'neighbouring' operation is of order $O(n^2)$, but during the subsequent computation of pair interactions, only the particles in the list have to be considered, which is a $O(n)$ operation. [15, 22]

When a particle has moved more than half the skin distance, the neighbour list should be rebuilt. Usually, it is checked periodically (every n timesteps) if this is the case.

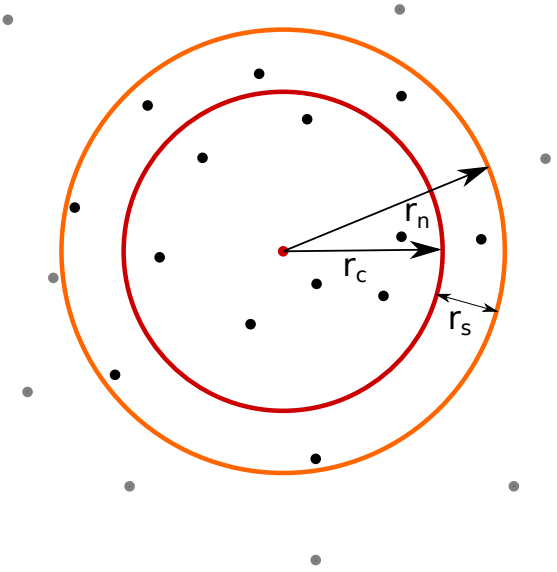

Figure 3.2: *Illustration of a neighbour list, showing a central particle with the cut-off radius r_c, the neighbour radius r_n and the skin distance r_s between them.*

There exists a trade-off in deciding upon appropriate values for the skin distance and the integration timestep. Choosing a small skin distance result in lists that are smaller so computation of pair interactions will be faster, but (depending on how much the particles move) the neighbour lists have to be rebuilt significantly more often which will negate that gain. Similarly for the timestep it applies that a larger timestep is more efficient but (because particles will move more in one timestep) neighbour lists have to be rebuilt more often and the skin distance might need to be increased to prevent that.

3.4 Statistical ensembles

When MD is carried out with typical integration schemes, the total energy in the system (the sum of potential and kinetic energy) is conserved. In other words, we will sample the microcanonical ensemble (NVE), since the number of particles and the total volume are constant as well. However, this is often times not desirable in practice, because this is not a realistic scenario in real-life experimental analogues, where the system is thermally coupled with the environment and heat exchange occurs to equalise the temperature of the system with that of the environment. The latter corresponds to a canonical ensemble (NVT). Here, the total energy is no longer conserved, since energy flows in and out of the system. Whereas a simulation of the microcanonical ensemble tends to maximise the entropy of the system, a simulation of the canonical ensemble tends to minimise the system's Helmholtz free energy. [18]

3.4.1 Thermostats

In order to sample the canonical ensemble using MD, we need to allow for some energy exchange mechanism that aims to keep the temperature of the system constant. There are several ways to accomplish this (called thermostats), but not all of them are computationally efficient and not all of them properly sample the canonical ensemble.

To compute the instantaneous temperature of a system in a MD simulation, the equipartition theorem, which says that every degree of freedom (d_f) contributes $\frac{1}{2}kT$ to the kinetic energy is invoked:

$$\frac{d_\mathrm{f}}{2}kT = \frac{1}{2}m\langle v^2\rangle. \tag{3.16}$$

In practice then, for an ensemble of particles with only translational (and no internal) degrees of freedom in three dimensions, the temperature can be calculated as:

$$T_\mathrm{k} = \frac{m}{3Nk}\sum_{i=1}^{N}|v_i|^2 \tag{3.17}$$

where T_k denotes the instantaneous temperature as calculated from the average kinetic energy of the system and N is the number of particles in the system.

Velocity scaling and Berendsen

The simplest way to accomplish thermostatting is velocity scaling. This means that at every timestep, the velocities for all particles in the system are multiplied by a factor so that after the scaling operation, the average temperature of the system is equalised with a setpoint (corresponding to the environment temperature). [23]

This factor can be calculated by considering the influence of the particle velocities on the instantaneous temperature T_k (Equation 3.17):

$$\begin{aligned} T_0 &= \frac{m}{3Nk}\sum_{i=1}^{N}(\lambda|v_i|)^2 \\ &= \lambda^2 T_\mathrm{k} \qquad (3.18) \\ \lambda &= \sqrt{T_0/T_\mathrm{k}} \qquad (3.19) \end{aligned}$$

where T_0 is the setpoint temperature and λ is the velocity scaling factor.

The main problem with this approach is that it does not allow for fluctuations in temperature at all: at every timestep, the temperature is exactly equal to the setpoint. This is obviously not realistic and does not correspond to the canonical ensemble. [23]

The Berendsen thermostat is a slight modification of the above method. Instead of rescaling the velocities to make the system correspond exactly to the setpoint temperature at every timestep, the scaling factor is chosen in such a way that the rate of change of temperature is proportional to the temperature difference. This is essentially Newton's law of cooling:

$$\frac{\mathrm{d}T}{\mathrm{d}t} = \frac{1}{\tau}(T_0 - T) \tag{3.20}$$

where τ is a time constant that can be understood as a parameter controlling the degree of thermal coupling between the system and its environment. The solution of this first-order differential equation is the well-known exponential decay of the system's temperature to the setpoint. The velocity scaling factor in this case can be derived by simply considering the desired new temperature T_{new} and plugging it in Equation 3.19 as T_0:

$$T_{\text{new}} = T_{\text{k}} + \frac{\delta t}{\tau}\left(T_0 - T_{\text{k}}\right) \tag{3.21}$$

$$\lambda = \sqrt{1 + \frac{\delta t}{\tau}\left(\frac{T_0}{T_{\text{k}} - 1}\right)} \tag{3.22}$$

Three limiting cases for the Berendsen thermostat can be identified. In the limit as $\tau \to \infty$ the heat flow approaches zero and the Berendsen thermostat is idle. The resulting MD simulation will effectively sample the microcanonical ensemble, just like when no thermostat is employed. When the time constant is equal to the MD timestep ($\tau = \Delta t$), the Berendsen thermostat reduces to simple velocity scaling. For values of $\tau < \Delta t$, the thermostat is unstable and the system temperature will oscillate. Still, even for values of $\Delta t < \tau < \infty$, the Berendsen thermostat does not formally generate the canonical ensemble. [24]

Naive velocity scaling and the Berendsen thermostat generate what is called an isokinetic ensemble. The average kinetic energy per particle in this case is as expected, but it does not obey the distribution prescribed by the canonical ensemble (the Maxwell-Boltzmann distribution, Equation 3.23) [15, 23]. As a consequence of this fact, this sometimes gives rise to strange effects, such as the so-called Flying Ice Cube Effect, in which the equipartition theorem is violated and kinetic energy from high-frequency modes drains in to low-frequency modes. The effect gains its name by the prime example where a fluid freezes over time in a simulation with a velocity scaling thermostat and all thermal kinetic energy is converted into zero-frequency translational energy [25, 26].

$$P(v) = \left(\frac{\beta}{2\pi m}\right)^{3/2} \exp\left(\frac{-\beta m v^2}{2}\right) \tag{3.23}$$

Andersen

The simplest thermostat that *does* produce a proper canonical ensemble is known as the Andersen thermostat. In this method, the system is coupled to a fictitious heat reservoir through occasional stochastic forces on randomly selected particles. The particles that are selected to 'undergo a collision' have their velocities reset, and get assigned new velocities drawn from the Maxwell-Boltzmann distribution (Equation 3.23) corresponding to the desired temperature. Collisions are uncorrelated and their frequency follows a Poisson distribution (Equation 3.24). The average frequency that a particle is selected to undergo collision is $\nu \delta t$. As such, this can be considered a Monte-Carlo (MC) process. [15, 23, 27]

$$P(t) = \nu \exp(-\nu t) \tag{3.24}$$

It can be shown that MD with the Andersen thermostat samples the true canonical ensemble. [15]

Langevin and Brownian dynamics

The Langevin thermostat models the (thermal) effects of an implicit solvent by adding, besides the conservative forces arising from force field interactions, a random force and a dissipative force to each particle at every timestep. The random force can be interpreted as Brownian motion: the stochastic collisions of the implicit solvent particles rapidly bouncing around due to their thermal motion, and similarly the dissipative force represents hydrodynamic (Stokes') drag between the particles (random collisions of solvent particles with the simulated particles) and the implicit solvent. [28, 29]

An MD simulation using the Langevin thermostat is also said to perform Langevin dynamics. Langevin dynamics can be regarded as a coarse-graining method since it effectively coarse-grains away fast modes of motion of the solvent. The entire solvent is replaced by an implicit solvent, while its net effect (Brownian motion) is retained. It thereby takes advantage of the separation in characteristic timescales between solvent and the explicitly simulated molecules. It goes without saying that this is only justified if the solvent molecules are much smaller, but this is unquestionably the case in the case of macromolecules or colloids. [30]

The Langevin equation (Equation 3.25) is a simple force balance that captures this idea. F_c refers to the sum of all conservative forces (forces arising from pair interactions), F_d is the dissipative Stokes' drag which is equal to $-\gamma v(t)$, with γ being the drag coefficient and $v(t)$ naturally the velocity at time t, and F_b is the random force due to Brownian motion.

$$\begin{aligned} m\frac{\mathrm{d}v(t)}{\mathrm{d}t} &= F_c + F_d \quad + F_b \\ m\frac{\mathrm{d}v(t)}{\mathrm{d}t} &= F_c - \gamma v(t) + F_b \end{aligned} \tag{3.25}$$

The random force adds kinetic energy to the system while the dissipative force removes it. The fluctuation-dissipation theorem (FDT, Equation 3.26) states that on average, the stochastic fluctuations are perfectly balanced by the dissipative forces so that in thermal equilibrium, the combined energy stays constant.

$$\langle F_d + F_b \rangle = 0 \tag{3.26}$$

When γ is very high, i.e. Brownian motion and viscous drag dominate over the normal motion of the particles, correlations in the velocity of a particle decay on a timescale shorter than over which the conservative forces change (i.e. the timescale of interest). The system is then called *overdamped*, and Langevin dynamics reduces to a special case called Brownian dynamics [28]. This can be obtained formally by realising that in that case, on average, the displacement of a particle is zero, so the Langevin equation transforms into:

$$
\begin{aligned}
m\frac{\mathrm{d}v(t)}{\mathrm{d}t} &= F_\mathrm{c} - \gamma v(t) + F_\mathrm{b} \\
F_\mathrm{c} - \gamma v(t) + F_\mathrm{b} &= 0 \\
\frac{\mathrm{d}r}{\mathrm{d}t} &= \frac{1}{\gamma}(F_\mathrm{c} + F_\mathrm{b}).
\end{aligned} \tag{3.27}
$$

A possibly unexpected benefit of the Langevin thermostat is that it improves the stability of the simulation. This means we can get away with a much larger timesteps than in the case of NVE dynamics. This makes it the thermostat of choice for the equilibration of especially polymer systems, which can have very long relaxation times. [23]

Dissipative particle dynamics

The Langevin thermostat is a fine choice when an implicit solvent is desired, but the problem with stochastic thermostats of the Andersen and Langevin kind is that while they correctly generate a canonical ensemble and thereby correctly reproduce time-independent properties, the *dynamic* properties of the system are generally disturbed by the unrealistic stochastic alterations to the particle velocities. For example, these thermostats are not suitable if one wants to study the diffusion rate in a system because momentum transport is destroyed.

Dissipative Particles Dynamics (DPD) is similar to Langevin dynamics, but only adds pairwise forces to particles. As such, momentum is preserved in a collision, even though the forces of the collisions are random. As such, DPD preserves momentum transport as it does not suffer from the problems that the other stochastic thermostats treated here suffer from. [23]

Since the random and dissipative forces are now pairwise, a cut-off is introduced just like in the case of the pair potentials. Without it, we would again obtain a n-body simulation. The force balance of an interaction between two particles within the cut-off distance now reads: [15]

$$
m\frac{\mathrm{d}v_\mathrm{i}(t)}{\mathrm{d}t} = F_\mathrm{c}(r_\mathrm{i,j}) - \gamma\omega(r_\mathrm{i,j})v_\mathrm{i,j}(t) + F_\mathrm{b}(r_\mathrm{i,j}) \tag{3.28}
$$

where $(r_\mathrm{i,j})$ is the distance between the two particles, $(v_\mathrm{i,j})$ is the relative speed between the two particles, and ω is a factor that describes the variation of the drag coefficient with the distance.

Nosé-Hoover

The Nosé-Hoover thermostat is a deterministic thermostat that properly samples the canonical ensemble. Its principle is based around an extended Lagrangian that samples that is setup in a way that samples the canonical ensemble in the real system. [15]

Synopsis

In Table 3.1, an overview of the above discussed thermostats is given. In general, stochastic thermostats destroy momentum transport and should thus be avoided for simulations where these properties are of interest, with the exception of DPD.

In this work, the Langevin thermostat is used regularly for equilibrating polymer systems. For production runs, it is switched for the Nosé-Hoover thermostat.

Table 3.1: *Overview of thermostats*

Thermostat	Short description	Generates proper NVT ensemble?	Conserves momentum transport?	Stochastic?
Velocity scaling	All velocities rescaled to give exact desired temperature every timestep	No	Yes	No
Berendsen	Exponentially-decayed velocity rescaling	No	Yes	No
Andersen	Hybrid MD/MC	Yes	No	Yes
Langevin	Stochastic collisions and Stokes dissipation	Yes	No	Yes
Nosé-Hoover	Extended Lagrangian	Yes	Yes	No
DPD	Pairwise Langevin	Yes	Yes	Yes

3.4.2 Chemostatting using Grand Canonical Monte-Carlo

In the previous section, we discussed how we can use thermostats to exchange energy with the environment with the goal to keep a constant temperature and sample the canonical ensemble. We can do something similar for the grand canonical ensemble, although this is less trivial in MD. It turns out that we need a kind of hybrid MD/MC approach to exchange particles with the environment, because in a strictly MD simulation, particles cannot inserted or removed from the system; in a MD simulation, a system is inherently closed.

To overcome this limitation, we can intersperse dynamics with Monte Carlo sweeps that insert or remove particles and keep their concentration or partial pressure (technically: chemical potential) constant. When this is done for only one type of particles, this way we can achieve a μVT ensemble for only those particles, while the rest of the system is simulated at constant NVT, as is customary for MD.

The simulation maintains a virtual reservoir (Figure 3.3), of which the chemical potential (μ) or pressure (of a virtual ideal gas) is imposed. During a GCMC sweep, the simulation code tries to exchange a number of particles between the virtual reservoir and (a region of) the real simulation domain. If the free energy change of an exchanged particle is negative (favourable), the exchange is accepted. If it is not, the exchange can still be accepted by random chance, which probability is determined by the system's partition function, reflecting the Boltzmann statistics in the grand canonical ensemble. [17]

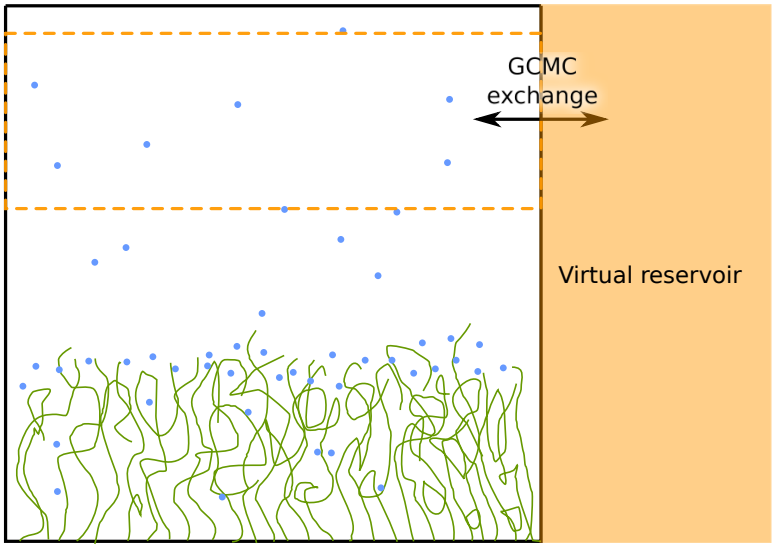

Figure 3.3: *Schematic illustration of the operation of the GCMC chemostat. The virtual reservoir is only a mathematical construct and does not spatially reside next to the box.*

3.5 Coarse-graining

Fully atomistic MD simulations, that is, simulations where the smallest features simulated are atoms, would be too computationally expensive for large-scale polymer systems. Because we are not interested in the behaviour of the system on that scale anyway, we use a technique called coarse-graining. This means that groups of atoms are replaced by a single larger unit, or bead, which reproduces the same large-scale phenomena as the fully atomistic simulation. Depending on the coarse-graining model, chemical specificity can be limited or even lost.

3.5.1 Kremer-Grest

The Kremer-Grest model [19] is a coarse-grained polymer model where polymers are represented by beads comprising several repeat units that are freely-joined and connected by springs. As such, it is a specific MD implementation of a bead-spring model of polymers. One bead corresponds approximately to one Kuhn unit. Chemical specificity is lost; with Kremer-Grest, one simulates a generic polymer in reduced Lennard-Jones units. Mappings exist that can be used to convert observed properties to real units for several examples of polymers afterwards.

In the Kremer-Grest model, non-bonded particles interact using a (truncated) LJ potential and bonded particles interact using a Finitely Extensible Non-linear Elastic (FENE) potential (Equation 3.31) combined with a WCA potential.

$$U_{\mathrm{FENE}}(r) = -0.5KR_0^2 \ln\left(1 - \left(\frac{r}{R_0}\right)^2\right) \tag{3.29}$$

$$U_{\mathrm{WCA}}(r) = \begin{cases} U_{\mathrm{LJ}}(r) + \epsilon & \text{for } r \leq 2^{1/6} \\ 0 & \text{for } r > 2^{1/6} \end{cases} \tag{3.30}$$

$$U_{\mathrm{bond}}(r) = U_{\mathrm{WCA}}(r) + U_{\mathrm{FENE}}(r) \tag{3.31}$$

In order to simulate an implicit solvent, a Langevin thermostat is employed, which reproduces the effects of Brownian motion on the particles by the implicit solvent.

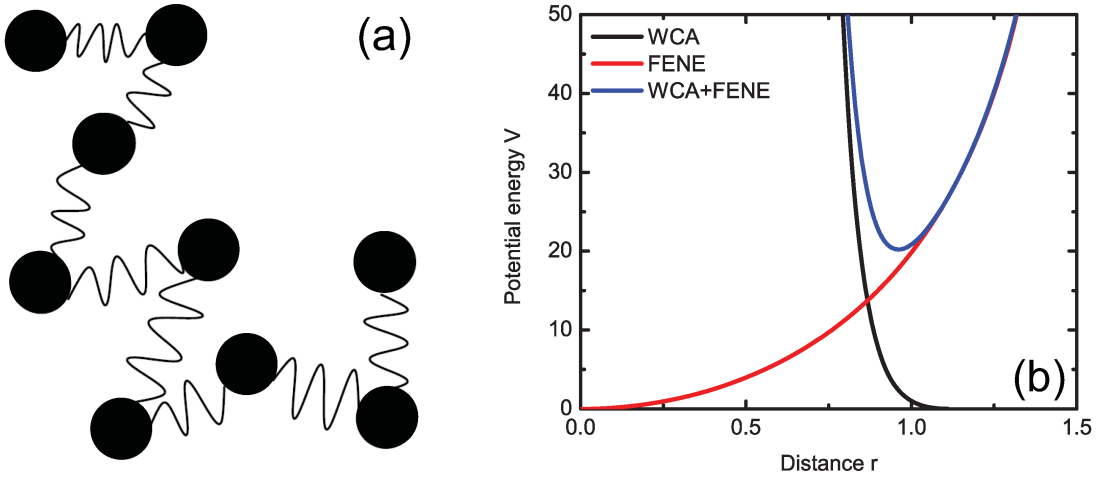

Figure 3.4: (***a***)*: Illustration of a bead-spring model of a polymer and the FENE+WCA potential* (***b***)*.* [31]

Simulations using the Kremer-Grest model are commonly performed using reduced LJ units defined on basis of the parameters of the LJ potential (σ and ϵ):

Table 3.2: *Reduced LJ units*

Length	$r* = r/\sigma$
Energy	$E* = E/\epsilon$
Time	$t* = t/\tau = t\sqrt{\epsilon/(m\sigma^2)}$
Temperature	$T* = k_B T/\epsilon$

3.5.2 MARTINI

The MARTINI force field [32] is a coarse-grained framework for modelling a wide range of molecules. It is primarily developed for biomolecules, most particularly lipids, but coarse-grained models for various polymers based on MARTINI have been developed as well. MARTINI is often used with the MD package GROMACS, but it can be implemented in other MD packages as well.

The force field consists of a description of how to map groups of atoms to coarse-grained beads, a pair potential for non-bonded interactions, a potential for bonded interactions, and an angle potential and (possibly) a dihedral potential for representing chain stiffness.

In the MARTINI framework, molecules are coarse-grained by mapping (on average) 4 heavy atoms (C, N, O, etc) and their associated hydrogen atoms to one MARTINI bead. The framework comes with a library of standard beads (building blocks), ranging from polar to apolar and varying in hydrogen bond capabilities and charge. By this approach, a wide range of molecules can be coarse-grained while still maintaining chemical specificity (see Figure 3.5). [33]

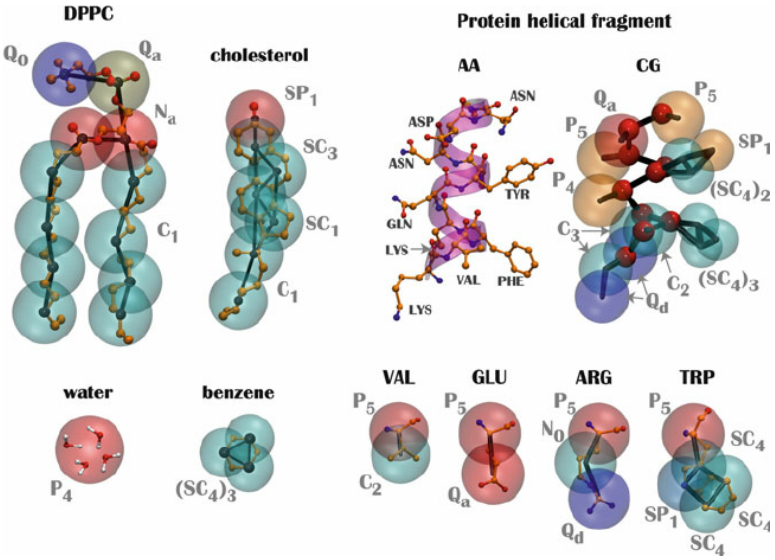

Figure 3.5: *Graphical explanation of the coarse-graining principle of MARTINI showing how atomistic models of phospholipids, proteins, water, benzene, and amino acids are mapped to MARTINI beads.* [34]

The non-bonded interactions are parametrised based on reproducing experimental properties (top-down), while bonded interactions are parametrised based on reproducing fully atomistic simulations (bottom-up).

The non-bonded interactions are described by the Lennard-Jones potential (Equation 3.10) truncated and smoothly shifted[1] at $1.2\,\mathrm{nm}$. The interaction parameters (ϵ and σ of the LJ potential) for every pair of MARTINI beads are specified by an interaction matrix. Charged particles additionally interact by an electrostatic (Coulomb) potential (Equation 3.32), also truncated at $1.2\,\mathrm{nm}$. The latter mimics the effective distance-dependent electrostatic screening. [32, 33]

$$U_{\mathrm{C}\,i,j} = \frac{q_i q_j}{4\pi\epsilon r_{i,j}} \tag{3.32}$$

Bonded interactions are described by the simple harmonic potential (Equation 3.33). Chain stiffness is described by the harmonic cosine potential (Equation 3.34) which gives the energy as function of the angle between 1-3 consecutive atoms. Similarly, a dihedral potential that gives the energy as function of the torsion angle between 1-4 consecutive atoms can be used. MARTINI does not specify a potential for this, but specific models based on MARTINI can implement one. [32, 33]

[1] The truncation scheme used is similar to LJ/spline discussed in 3.2.1 and implemented as `lj/gromacs` in LAMMPS.

$$U_{\text{bond}}(r) = \frac{1}{2} K_{\text{bond}}(r - r_0)^2 \tag{3.33}$$

$$U_{\text{angle}}(\theta) = \frac{1}{2} K_{\text{angle}}(\cos(\theta) - \cos(\theta_0))^2 \tag{3.34}$$

4 | Simulations

4.1 Model and methods

All simulations were performed using the MD package LAMMPS (Large-scale Atomic/-Molecular Massively Parallel Simulator) [35]. Values are specified in reduced Lennard-Jones units by default, unless a (dimensioned) unit is specified.

4.1.1 Lennard-Jones vapour-liquid equilibria

In order to study vapour hydration of polymer brushes, solvent is kept in the vapour phase above the brush at a constant pressure. First, an appropriate value for this pressure must be established. We want the pressure to be reasonably close to, but not higher than the saturation pressure, as it would condense in the latter case. We can quantify this using the *relative vapour pressure*, which we define here as $\frac{p}{p_{\mathrm{sat}}}$. Note that if the solvent would be water, this term would be identical to *relative humidity*. Nevertheless, we use the term relative vapour pressure here because our model is not specific to water. Our goal here is to keep the solvent at $\frac{p}{p_{\mathrm{sat}}} \approx 0.75$. To know what pressure we need to use, we first need to determine p_{sat}.

The solvent consists of Lennard-Jones fluid (Lennard-Jonesium). The caveat is that the thermodynamic properties of a LJ fluid are very sensitive to the exact truncation and shifting method used [36]. Therefore, it is necessary to first determine the saturation pressure of the LJ fluid at the conditions as it exists in our simulations before we can proceed with the polymer brush simulations.

For a vapour in equilibrium with its liquid phase, the pressure should be constant and equal to the equilibrium vapour pressure (saturation pressure) of the fluid. Hence, as long as the system's overall density is in the coexistence region, its pressure should be independent of the overall density. If one would increase the density of a two-phase system in equilibrium, either by decreasing the system's volume or by adding particles, more particles condense into a liquid and the ratio of particles in the liquid phase over particles in the vapour phase increases but the pressure stays exactly constant. When the overall density equals the density of the liquid phase, the entire system is in the liquid phase and we have moved out of the coexistence region. Beyond this point, pressure increases with increasing density again because we are then compressing the liquid.

However, it is not so trivial to simulate vapour-liquid coexistence and measure the saturation pressure using MD as one might be inclined to think. First of all, in many systems phase separation is suppressed at MD length and time scales, but this can be easily overcome by choosing a sufficiently large box size. The second problem is that we cannot simply use the total (mean) pressure in the system because this includes (significant) contributions of interfacial tension. We can solve this by sampling pressure *profiles* and looking specifically for the pressure within the bulk of each phase.

A cubic, periodic simulation box containing $N = 10000$ LJ particles is set up with a number density of ρ. It follows that the lengths of the box are:

$$L = \left(\frac{N}{\rho}\right)^{1/3}. \tag{4.1}$$

The temperature is kept constant at $T = 0.85$ using a Nosé-Hoover thermostat with a time constant of $0.5\ \tau$.

The simulation is run for 1M timesteps with a timestep of $\Delta t = 0.005\ \tau$ while pressure profiles over z are sampled.

A set of 8 simulations is run with varying densities in the coexistence region: $\rho = 0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5$.

4.1.2 Kremer-Grest polymer brush vapour solvation

Now that we know the saturation pressure of the LJ fluid, we can proceed to the simulations of the polymer. A rough sketch of the system is shown in Figure 4.1. A polymer brush is grafted to a planar wall at the bottom of the box ($z = 0$) with solvent vapour above it. In a region at the top of the box, the solvent is allowed to enter and leave the system through the GCMC mechanism outlined in 3.4.2.

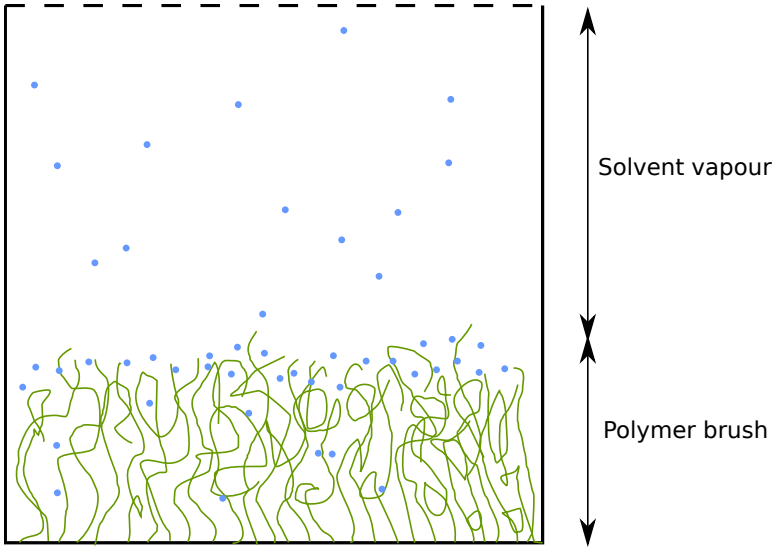

Figure 4.1: *Schematic illustration of the simulated system.*

The self-affinity of the polymer, the polymer-solvent affinity, as well as the solvent pressure together determine the sorption behaviour of the solvated brush system. The goal of these simulations is to investigate the behaviour of the brush system as a function of these parameters and identify distinct regimes.

All units in the Kremer-Grest simulations are reduced LJ units (Table 3.2), and the timestep used is $\Delta t = 0.005\ \tau$.

A system was set up consisting of a rectangular box of dimensions 31 x 35 x 46 σ (x, y, z respectively) that is periodic in x and y. A polymer brush is created by attaching chains with lengths of $N = 30$ in a random fashion to a flat stationary wall at $z = 0$ consisting of LJ particles arranged in a hexagonal close-packed lattice. The wall is rigid and fixed in place. The grafting density is $0.34\ \sigma^{-2}$. This value was chosen so that the system exists in a brush regime for all used solvent affinities.

All non-bonded interactions are described by the potential-shifted LJ potential with $\sigma = 1$ and $r_c = 2.5\ \sigma$ unless otherwise stated. Solvent self-interactions use $\epsilon_{ss} = 1$ while polymer-solvent interaction energies (ϵ_{ps}) and polymer self-interaction energies (ϵ_{pp}) are varied. Interactions with the wall are cut-off at $r_c = 2^{1/6}\ \sigma$, resulting in the WCA potential and ensuring purely repulsive interactions of both polymer and solvent with the wall.

Simulation procedure

The polymer system is first equilibrated from a fully-stretched initial situation for 10^6 timesteps. Then, solvent particles are introduced by the GCMC mechanism, which attempts 500 insertions/deletions every 10^4 timesteps. This interval is sufficiently shorter than the relaxation times of sorption in the brush. This simulation is run for $9 \cdot 10^6$ timesteps ($4.5 \cdot 10^4\ \tau$).

Both the polymer and solvent are kept at a constant temperature of $T = 0.85$ using a chained Nosé-Hoover thermostat with a time constant of $5\ \tau$. In combination with the GCMC chemostat, this yields a grand canonical (μVT) ensemble for the solvent, and a canonical (NVT) ensemble for the polymer.

Density profiles of the polymer and solvent over the z direction are output.

4.1.3 MARTINI polymer brush vapour solvation

The Kremer-Grest simulations discussed above predict interesting results, but are chemically aspecific. The affinity between the polymer and the solvent and the self-affinities of the polymer and solvent have been chosen freely and are possibly not realistic.

The goal of performing MARTINI simulations is to identify 'real' combinations of polymers and solvents for which we reproduce the sorption regimes found by the Kremer-Grest simulations.

As MARTINI is chemically-specific, a MARTINI model that parametrises a specific polymer is required. As a starting point, poly(ethylene) (PE) with acetone vapour is evaluated. Water vapour was also considered, but was deemed computationally infeasible because of its low volatility and thereby scarce vapour density.

A new script was developed for generating the initial system (see A.2 for details). Unlike the Kremer-Grest system which uses a flat stationary wall consisting of LJ particles arranged in a hexagonal close-packed lattice to which the 'grafting' beads of the polymer are bonded, the new generator does not use such a explicit structured wall. Instead, the grafting beads are made stationary by fixing them in place and a flat structureless mathematical wall which exerts a purely repulsive harmonic force in the normal direction only is put in place.

Determination of saturation pressure

As with the Kremer-Grest simulations (4.1.1), as a first step, the saturation pressure of the solvent was determined using liquid-vapour coexistence simulations. This time, a tetragonal box with an aspect ratio of 5 was used instead of a cubic box, in order to facilitate the slab geometry of the system at lower densities. Vapour-liquid simulations are run with varying densities of MARTINI $\mathrm{N_a}$ beads in the coexistence region: $\rho = 0.1, 0.2, 0.3, 0.4, 0.5\ \text{Å}^{-3}$. The simulations are run for 5M timesteps with a timestep of $\Delta t = 10\,\text{fs}$.

PE

For PE, the model by Panizon et al [37] is used. Here, a PE chain is coarse-grained as a chain of MARTINI $\mathrm{C_1}$ beads, where a $\mathrm{C_1}$ bead represents two ethylene repeat units, i.e. four carbon atoms and associated hydrogens (as is usual with MARTINI). To represent the structural properties of PE, consecutive beads are constrained by angle and dihedral potentials.

Matching the Kremer-Grest simulations

The MARTINI PE simulations are matched with the chemically aspecific Kremer-Grest simulations using the mapping of the MARTINI model to the real polymer, and the mapping of the Kremer-Grest model to that same real polymer (as described in Table 3 in Kremer and Grest [19]): In the Kremer-Grest model, one bead corresponds to 2.76 ethylene units, that is, a Kremer-Grest bead is 1.38 as large as a MARTINI bead for PE. Also, $1\ \sigma = 5.1$ Å, $1\ \tau = 66\,\text{ps}$, and $1\ \epsilon/k_B = 448$ K [19].

To match the Kremer-Grest simulations, a grafting density of $0.34/5.1^2 \approx 0.013\ \text{Å}^{-2}$ is used. The chain length is chosen as 41 following the bead mapping relation described above ($30 \cdot 2.76/2 \approx 41$). The box is 200 Å wide in x and y and 225 Å high in z. As the polymer chain is $41 \cdot 4.7 = 192.7$ Å long at a fully extended state at equilibrium bond lengths, this is sufficiently large to prevent periodic artifacts from polymer chains interacting with their own image.

It is worth noting that the Kremer-Grest mapping is based on dynamic properties of the coarse-grained polymer, and that the model thus does not necessarily accurately reproduces all static properties like Kuhn length [19]. As the MARTINI model is more sophisticated, it is expected to perform better in this regard.

Simulation procedure

The polymer system is first equilibrated for 5000 timesteps by running dynamics with a limit imposed on the maximum movement of a particle in one timestep (`fix nve/limit`) of 1 Å and thermostatted by Langevin dynamics with a damping parameter of $5000\,\mathrm{fs}$. This is followed by an energy minimisation using the conjugate gradient method (`min_style cg`) and then again 1000 timesteps of more viscous Langevin dynamics with a damping parameter of $500\,\mathrm{fs}$ and without the limit. This procedure is chosen to relax the system from the low-entropy initial state (fully-extended chains) as quickly and efficiently as possible. The first Langevin dynamics run intentionally has a rather long damping constant in order not to cool the system too quickly; if that happens, the system will 'freeze' and take very long to relax to its equilibrium state. The longer damping constant allows the system to first heat up and stay at a relatively high temperature while it relaxes to 'anneal' the system. The second dynamics run has a much more viscous Langevin thermostat in order to drain the energy and cool the system to the desired temperature.

After the equilibration, solvent particles are introduced into the system in the same way as with the Kremer-Grest simulations. The simulation is run for $5 \cdot 10^9$ timesteps $(0.1\,\mathrm{ms})$. Density profiles are output.

Timestep and numerical stability

Generally an as long as possible timestep is desired for computational efficiency reasons. For MARTINI simulations, timesteps of around $20\,\mathrm{fs}$ are common. In some cases, a timestep as high as $40\,\mathrm{fs}$ is attainable, although this cannot always be reached due to limited numerical stability depending on the exact system [32]. In the case of the PE model, Panizon et al mention a shorter timestep of around $10\,\mathrm{fs}$ to $15\,\mathrm{fs}$ might be required to keep the simulation stable because the use of dihedral constraints in combination with an angle constraint with an equilibrium angle of $0°$, which results in numerical instabilities as the dihedral potential assumes asymptotic values when the relevant bond angles approach $0°$ [37].

We were not able to reproduce a numerical stable simulation with a timestep of $15\,\mathrm{fs}$. In fact, a timestep of $5\,\mathrm{fs}$ was required for the simulation not to crash and the total energy was still not conserved in NVE dynamics at a timestep as short as $1\,\mathrm{fs}$.

Using the rRESPA multi time scale integrator, stable dynamics was achieved with an outer timestep of $20\,\mathrm{fs}$ and a 10-fold as short inner timestep. Non-bonded pair interactions are evaluated on the outer timestep, while bond, angle, and dihedral interactions are evaluated on the inner timestep.

4.2 Results and discussion

4.2.1 Determination of saturation pressure

On the geometry

The system phase separates and adopts a slab geometry, i.e. there is layering in z with xy planes as interfaces (see Figure 4.2).

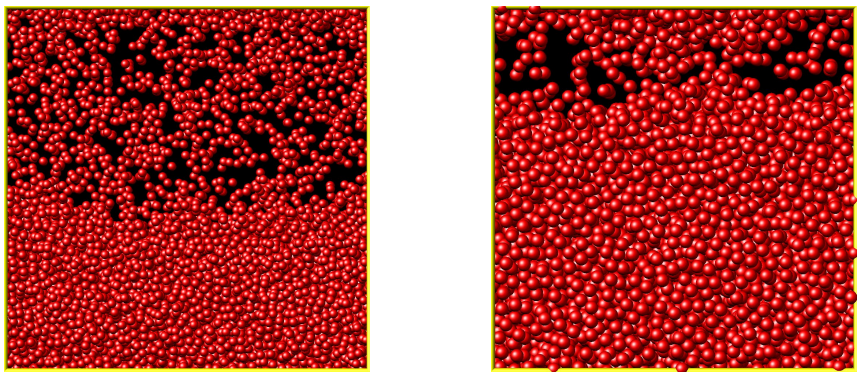

Figure 4.2: *Snapshots at* $\rho = 0.15$ (***left***) *and* $\rho = 0.5$ (***right***).

The interfacial tension dictates the geometry of the system. Depending on the overall density, there can be one or two stable situations: a droplet of liquid, pulled into a sphere by interfacial tension, surrounded by vapour, or a slab of liquid with vapour below and above it. The latter is only stable because it can 'cheat' by the periodic boundary conditions: there is no interface in two of the three dimensions.

In Figure 4.3 the interfacial areas are shown for the two situations depending on the overall density. Like in the simulations, the number of particles is kept constant here, so the system volume is varied. Only the liquid phase is considered here for simplicity reasons, i.e. there is no vapour phase and the number of particles in the liquid phase does not decrease when the system volume increases. This suffices for this purpose. It follows from this that the interfacial area for a droplet is constant and given by the surface area of a sphere of volume $\frac{N}{\rho_l}$ (Equation 4.2). The interfacial area of the slab is given by two times a face of the cubic box (Equation 4.3), which increases with increasing box volume (and thus with decreasing overall density).

$$A_{\text{droplet}} = \pi^{\frac{1}{3}} \left(6\frac{N}{\rho_l}\right)^{\frac{2}{3}} \tag{4.2}$$

$$A_{\text{slab}} = 2\left(\frac{N}{\rho}\right)^{\frac{2}{3}} \tag{4.3}$$

We can identify a couple of 'critical' densities, plotted in Figure 4.3 as vertical dashed lines. The first is the equilibrium density, where the interfacial areas of both geometries are equal. Hence, below the equilibrium density, a spherical droplet is favoured but a slab can sometimes still exist in a metastable state. The other is the critical droplet density, above which a spherical droplet simply cannot exist, not even in a metastable state, because its diameter exceeds the box dimensions. From this we can conclude we can use the overall density range from 0.19 to 0.7.

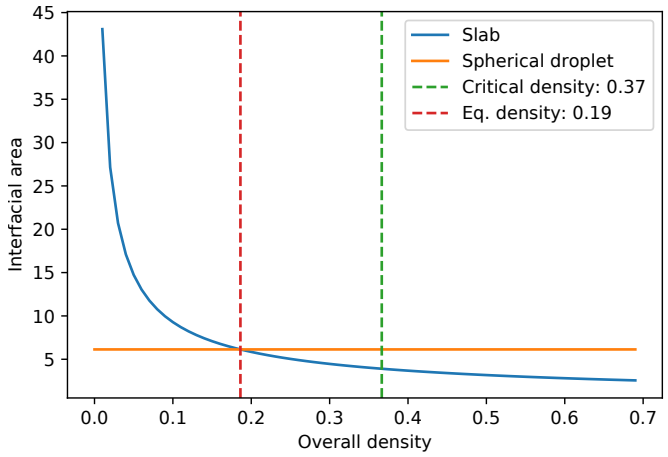

Figure 4.3: *Interfacial areas for spherical and slab geometries as function of the overall density for the LJ system.*

Even though there is interfacial tension, there is no meniscus because there are no walls; the system is periodic in all dimensions.

Density and pressure profiles

In Figure 4.4 an example of a density and pressure profile is shown for $\rho = 0.3$. Along the density profile, the gradient of the density is plotted. The gradient is used to identify the bulk phases: all points where the gradient of the density is lower than a certain threshold are marked as 'bulk' and the pressure at these points is averaged.

It can be seen that the pressure in the bulk is constant and that the bulk vapour pressure is equal to the bulk liquid pressure, as it should be. On the interfaces however, the pressure deviates and even goes into the negatives because of the contribution of the interfacial pressure.

Saturation pressure versus overall density

The construction described above is repeated for every ρ, so that the saturation pressure for every ρ is obtained. This is plotted in Figure 4.5. The data shows almost no dependency on the overall density, as we expected. The first point at $\rho = 0.15$ is considered an outlier because the interface here was not stable and planar enough and there was not enough bulk liquid in this case to get an accurate average bulk pressure.

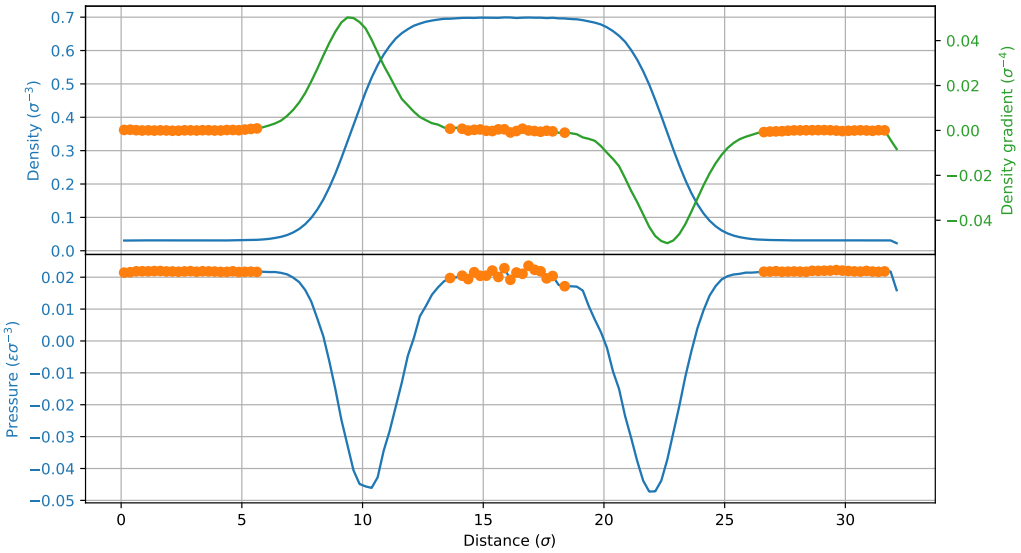

Figure 4.4: *Density (top) and pressure (bottom) profiles for $\rho = 0.3$. The orange points indicate the bulk phases.*

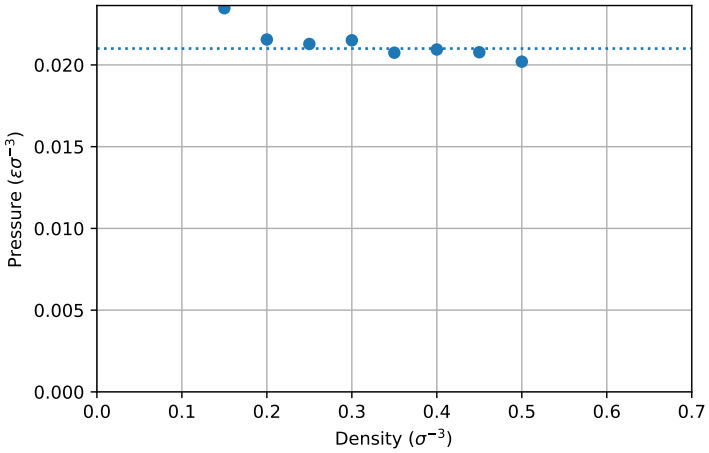

Figure 4.5: *Mean pressure in the bulk of both phases as a function of overall density.*

The final result of this simulations is that the saturation pressure of the Lennard-Jones fluid at $T = 0.85$ is $0.021\ \epsilon\sigma^{-3}$.

4.2.2 Kremer-Grest

Analysis of a density profile

The sorption behaviour of the brush system is evaluated by analysing time-averaged density profiles of the polymer and solvent over the z direction. In Figure 4.6 three examples of snapshots of vapour-solvated Kremer-Grest polymer brush system are shown with the corresponding density profiles (rotated 90°) below them.

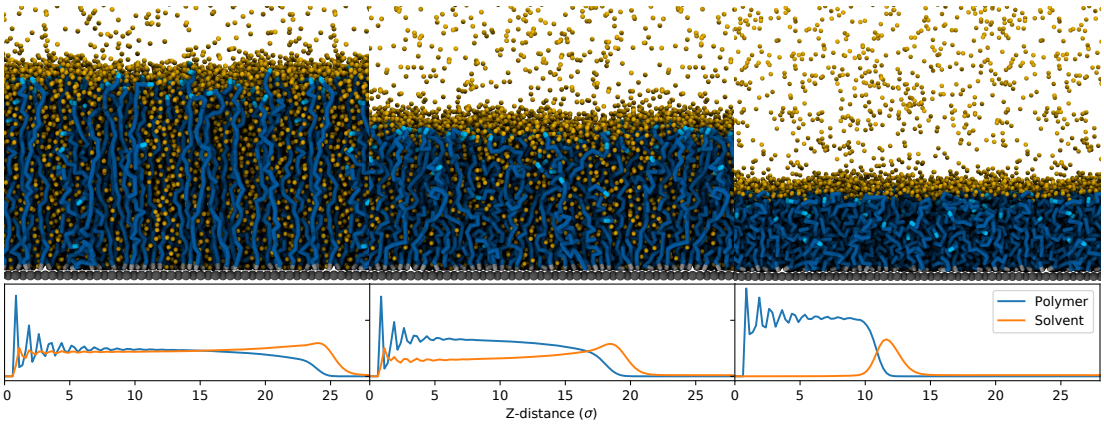

Figure 4.6: *Three snapshots at* $\epsilon_{\mathrm{ps}} = 1.40$ *for* $\epsilon_{\mathrm{pp}} = 0.60, 1.40, 2.00$ *(from left to right) with corresponding polymer and solvent density profiles.*

The oscillations in the polymer density close to the substrate ($z = 0$) are caused by layering effects stemming from the perfectly flat substrate. This is a well-known effect that also occurs in real-life systems [38, 39, 40], but is otherwise irrelevant for the scope of this work.

In Figure 4.7, a characteristic example of a density profile is shown with three dotted vertical lines that indicate distances significant for analysis of these profiles. The first is the inflection point (point of maximal slope) in the polymer density profile, which is used to define the border of the brush. There is still appreciable polymer density above this point, but the inflection point is commonly used as a mathematically significant definition of the *brush height* that does not depend on any 'magic' (arbitrary) numbers. When a more extensive measure of the brush height is desired (*outer brush height*), the point of maximum curvature of the polymer density profile is used. For determining the *border of the adsorption layer*, simply an arbitrary threshold in the gradient of the solvent density profile is used. Above this last border, the vapour bulk starts. The methods above are heuristic to a certain extent but were found to work well with the entire range of Kremer-Grest results.

These points are used to define limits used in the processing of the profiles. The solvent density profile is integrated up to the brush height to yield the amount of solvent *in* the brush (absorption), and likewise from the brush height up to the adsorption layer border to give the amount of solvent *on* the brush (adsorption). The *adsorption layer thickness* is measured from the outer brush height to give a more conservative measure of the thickness.

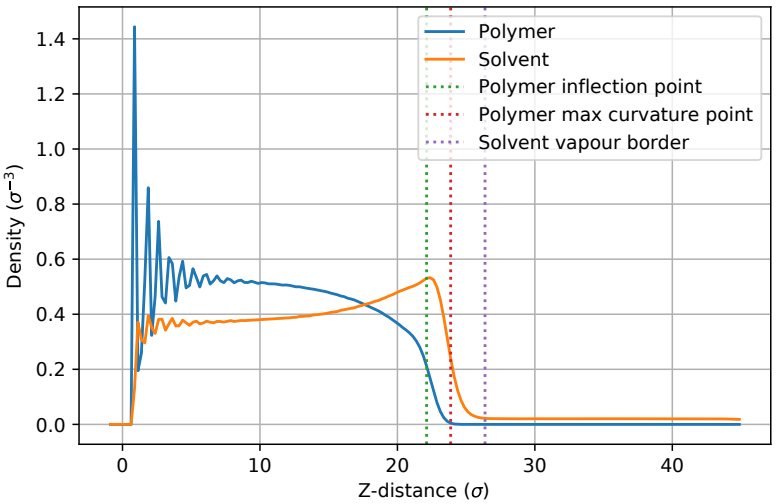

Figure 4.7: *Example of polymer and solvent density profiles with significant points indicated.*

Probing ϵ_{pp} and ϵ_{ps} parameter space

A parameter sweep over both the polymer self-affinity (ϵ_{pp}) and polymer-solvent affinity (ϵ_{ps}) was performed to obtain the grid in Figure 4.8. Each of the 16 plots shows the density profiles (number density vs z-distance) of the polymer and solvent in the brush system. The brush height is represented by the leftmost dotted line, and the rightmost dotted line indicates the adsorption layer border. Several sorption regimes can be distinguished: strong absorption in the top-left corner of the grid (low ϵ_{pp}, high ϵ_{ps}), only adsorption in the top-right corner (high ϵ_{pp}, high ϵ_{ps}), and no sorption at the bottom (low ϵ_{ps}). Note that adsorption occurs in every case there is sorption, as evidenced by the solvent density profile extending beyond the brush.

While the grid of density profiles contains all information about the sorption behaviour of the system, it can be hard to identify trends. For that reason, the amount of absorption and adsorption (as calculated by the integrals of the solvent density profile) can also be plotted as a heatmap (Figure 4.9). These heatmaps also have increased resolution in ϵ_{ps}. From this it is more clear that the absorption regime occurs in the top-left corner of the heatmap, approximately above a diagonal line. This suggests that there is a more fundamental parameter governing the absorption transition. Concerning the adsorption, it is evident that polymer self-affinity has negligible influence.

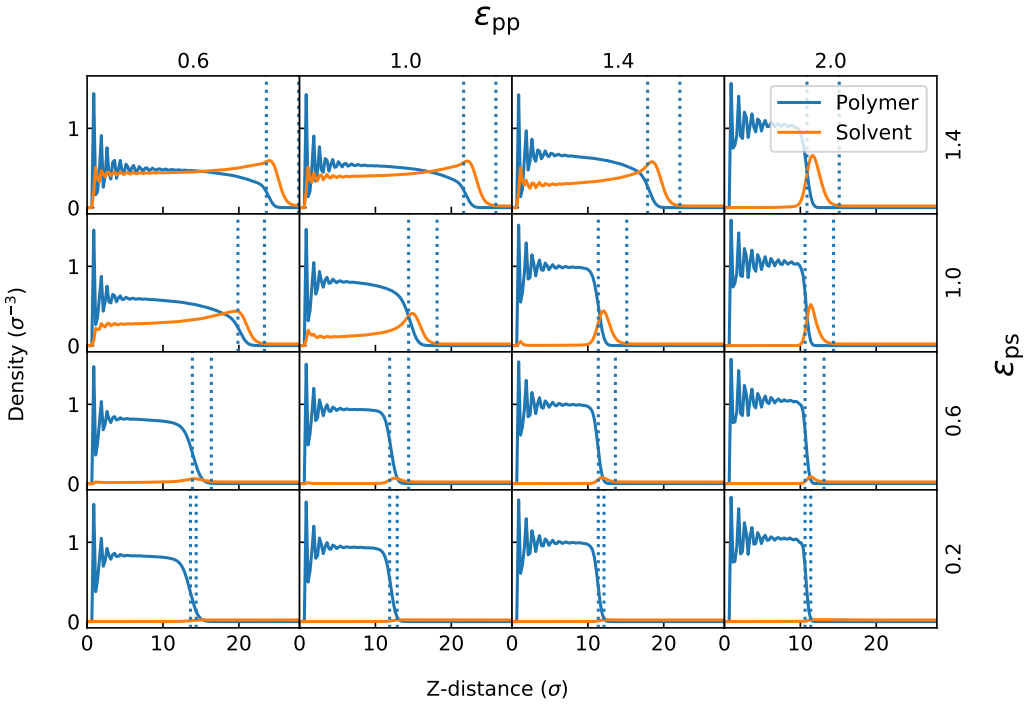

Figure 4.8: *Polymer and solvent density profiles for a 4x4 grid of ϵ_{pp} and ϵ_{ps} values. The dotted lines indicate the borders of the adsorption layer.*

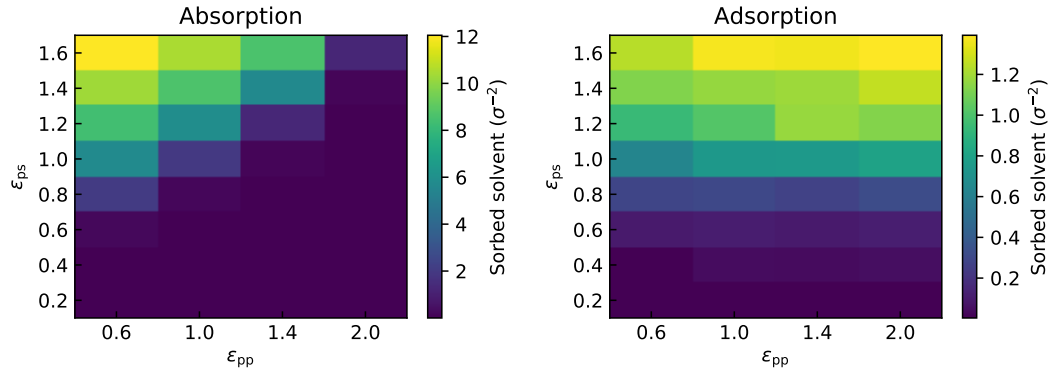

Figure 4.9: *Heatmaps of amount of absorbed (**left**) and adsorbed (**right**) solvent for a 8x4 grid of ϵ_{pp} and ϵ_{ps} values.*

Yet another representation of the same data is given in Figure 4.10 in the form of sorption curves as function of ϵ_{ps} for every ϵ_{pp}. The absorption curves have roughly the same shape, but are shifted horizontally. The adsorption curves overlap virtually perfectly for low polymer-solvent affinities, but diverge slightly as polymer-solvent affinities increase.

Our suspicion is confirmed in Figure 4.11, where the ϵ_{pp} and ϵ_{ps} parameters are remapped to the *relative affinity*:

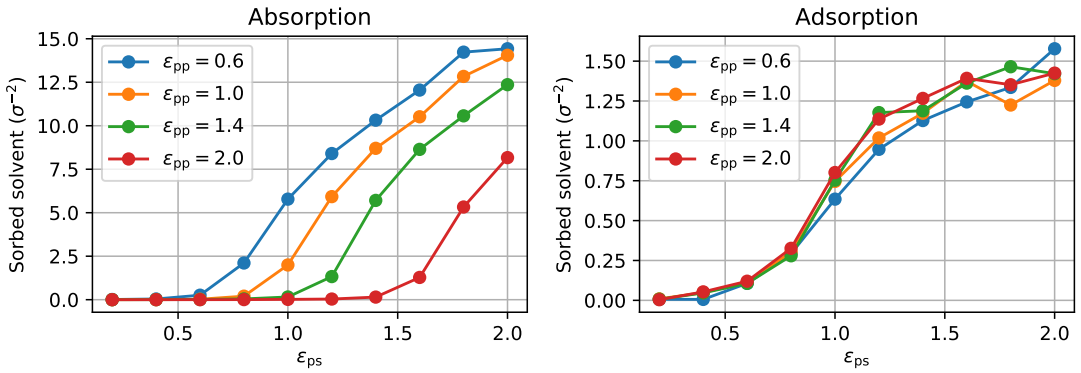

Figure 4.10: *Absorption (**left**) and adsorption (**right**) plotted against ϵ_{ps} for several values of ϵ_{pp}. The lines connecting markers are meant to guide the eye.*

$$W = (\epsilon_{pp} + \epsilon_{ss} - 2\epsilon_{ps})/2 \tag{4.4}$$

where ϵ_{ss} is the solvent self-affinity, fixed at 1 in all simulations. When the absorbed solvent is plotted against W, all the curves except for the case of highest polymer self-affinity perfectly overlap. This suggests that there is only one parameter, not two, controlling the absorption behaviour of the system. As W is purely enthalpic in nature, this also suggests that entropic contributions of the polymer stretching do not play a significant role. Furthermore, there seems to exist a (second-order) transition in the absorption, occurring at $W = 0$. This corresponds to the condition $\epsilon_{ps} = \frac{1}{2}\epsilon_{pp} + \frac{1}{2}$, which coincides with the slope we found in Figure 4.9.

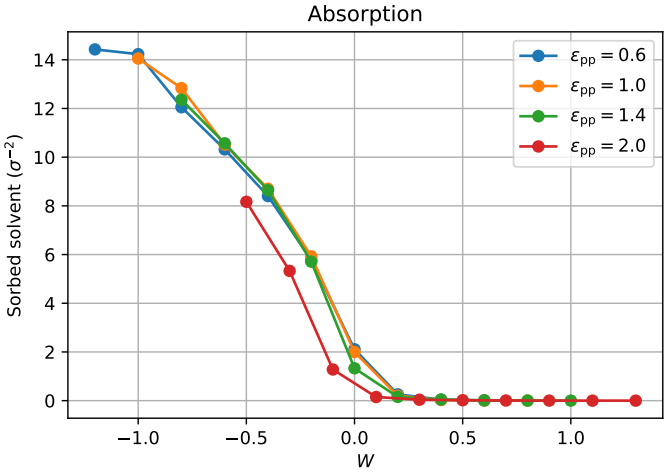

Figure 4.11: *Absorption plotted against W (the relative affinity) for several values of ϵ_{pp}. The lines connecting markers are meant to guide the eye.*

Varying solvent pressure

As we vary the solvent vapour pressure above the brush for two cases of polymer self-affinity ($\epsilon_{pp} = 0.6$ and $\epsilon_{pp} = 1.4$) we obtain the density profiles in Figure 4.12.

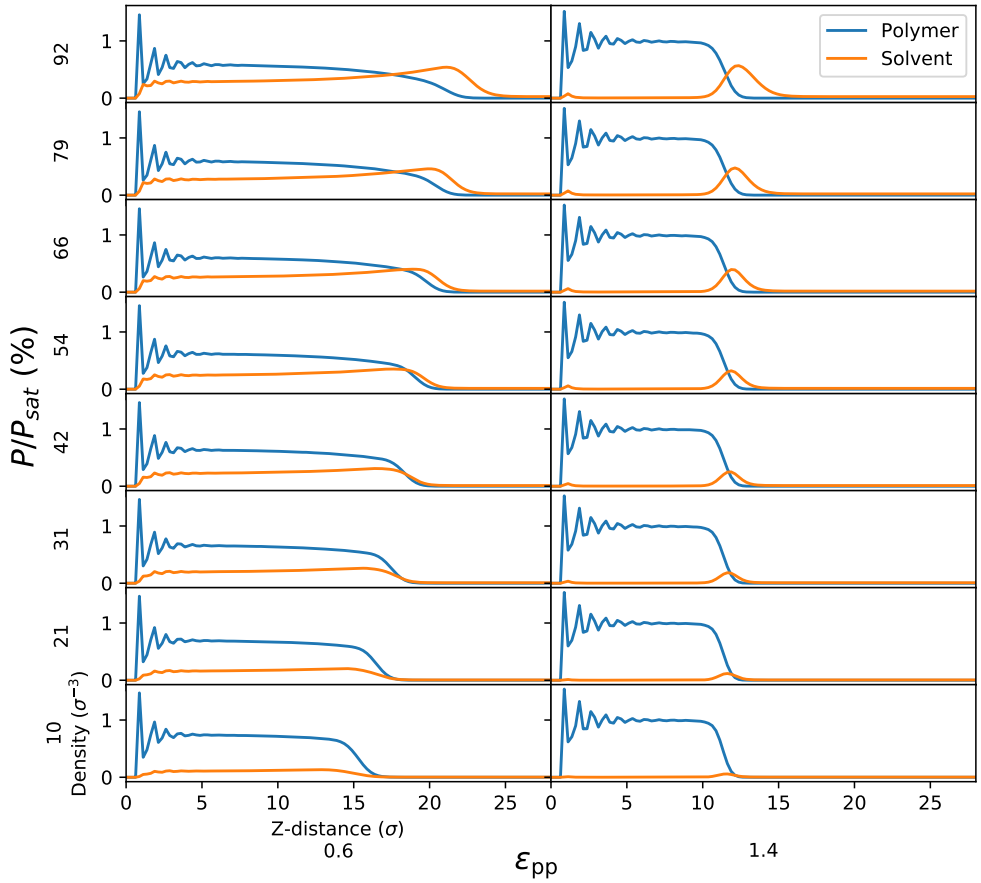

Figure 4.12: *Polymer and solvent density profiles for varied solvent pressure at* $\epsilon_{ps} = 1.0$ *and two* ϵ_{pp} *values* (*0.6 and 1.4*).

In a similar fashion as with the affinity parameter space results, these density profiles can be converted to absorbed and adsorbed solvent. These plots are shown in Figure 4.13. From there, it can be seen that the amount of absorbed solvent increases monotonically with relative solvent pressure. The adsorption behaviour is virtually identical for both cases of polymer self-affinity and it is purely the absorption behaviour which differs. The absorption also drives the brush swelling, as the brush height stays perfectly constant for the high polymer self-affinity case.

Also striking is that the brush height increases very linearly with the amount of absorbed solvent (Figure 4.14). This implies that no filling of free volume in the polymer brush by solvent occurs, since every added particle of solvent increases the total volume of the brush.

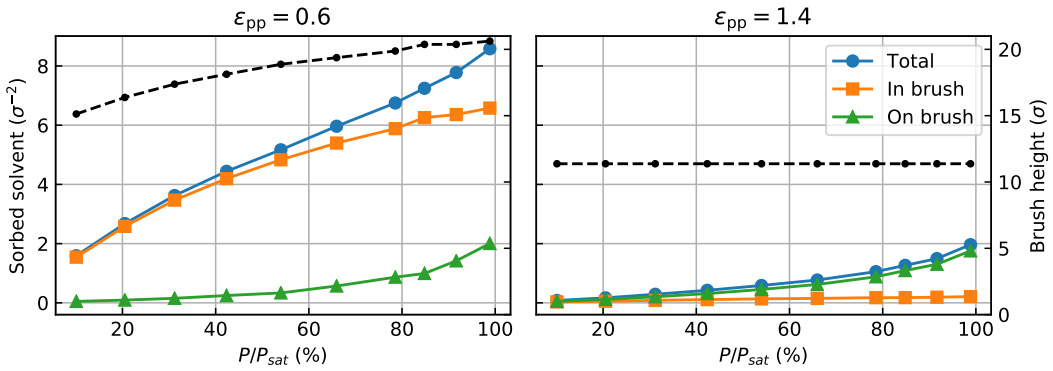

Figure 4.13: *Absorption, adsoption, and total sorption plotted against solvent pressure at* $\epsilon_{\mathrm{ps}} = 1.0$ *and two* ϵ_{pp} *values (0.6,* ***left*** *and 1.4,* ***right****). The dashed lines indicate brush height (right axis). The lines connecting markers are meant to guide the eye.*

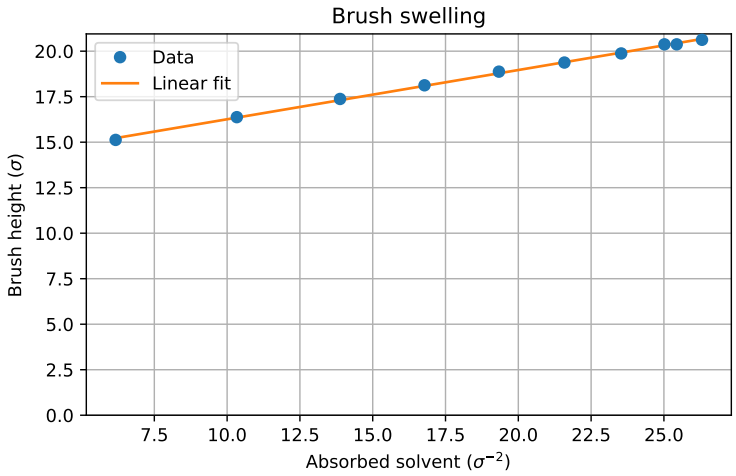

Figure 4.14: *Brush swelling: brush height plotted against absorbed solvent shows a linear relation.*

Adsorption layer thickness

As interface effects can be very sensitive to the truncation of the potentials, the effect of the polymer-solvent cut-off ($r_{\mathrm{c,\ ps}}$) on the thickness of the adsorption layer (measured from the outer brush height to the border with the bulk vapour) was investigated at two values for the solvent pressure: $p/p_{\mathrm{sat}} = 73.3\%$ and $p/p_{\mathrm{sat}} = 98.9\%$. In both cases, the adsorption layer thickness increases monotonically with the cut-off, but both plateau at a cut-off around $2.5\ \sigma$, which is reassuring for the validity of our simulations.

In both cases, the thickness of the adsorption layer exceeds the cut-off, something that was also already seen in Figure 4.12. This seems paradoxical at first, since solvent particles that are further away from the brush surface than r_{c} do not interact with the brush. However, this effect also occurs in solvent adsorption on a planar substrate (Figure 4.16).

This multi-layer adsorption in excess of the cut-off occurs because the outer solvent layers are attracted to the local solvent density enhancement of the layer beneath, which in turn is caused by the attractive interaction with the polymer brush.

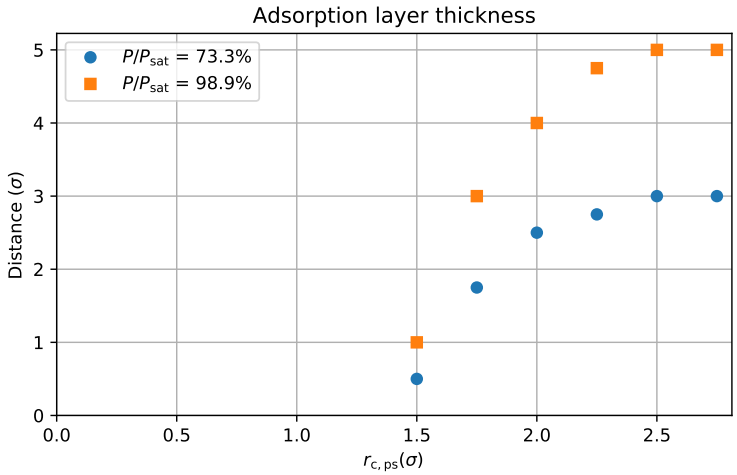

Figure 4.15: *Influence of the polymer-solvent cut-off on the adsorption layer thickness.*

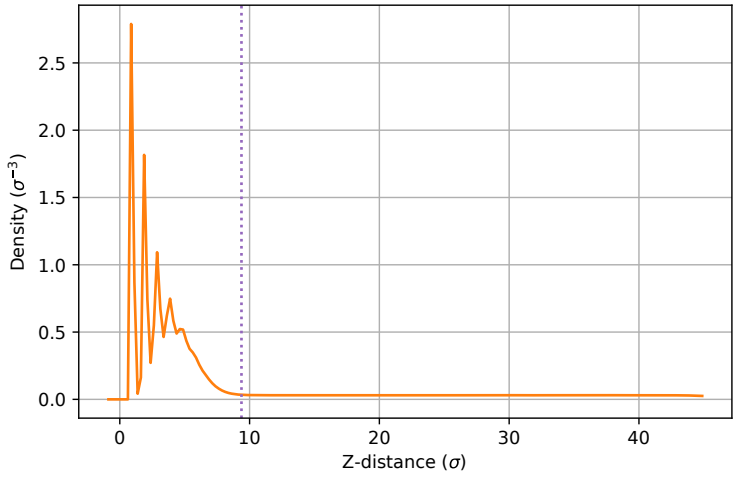

Figure 4.16: *Solvent density profile of adsorption on a planar substrate with a default cut-off of* $r_c = 2.5\ \sigma$ *at* $p/p_{sat} = 98.9\%$*, showcasing multi-layer adsorption with a thickness in excess of* r_c*.*

Solvent-induced crystallisation

When the polymer-solvent affinity was increased to extreme values, regular ordering of the polymer brush into 'pillars' with solvent between them was discovered (Figure 4.17). This system appears to be genuinely crystalline, as evidenced by xy (averaged over z) density maps and (the magnitude of) their Discrete Fourier Transforms (DFTs) (Figure 4.18). The DFT transforms a signal to the frequency domain, with points lying farther from the origin denoting higher frequency components of the signal. It helps to discern periodic patterns, although it is rather clear here from the density maps itself that the polymer is ordered in a crystal lattice. Still, it resolves that both the polymer and solvent are arranged in the same lattice since their DFT maps are virtually identical; the solvent just has a different basis.

Although polymer crystallisation is nothing particularly exciting, this result is because in this system the polymer does not crystallise by itself. Rather, the crystallisation is driven by the strong attractive interactions between the polymer-solvent binary system. Presumably, a polymer system like this does not exist in the real world because of the extreme interactions between polymer and solvent. A real-world example of a system that might come closest to the behaviour seen here might be a ionic liquid, but that is obviously not a polymer.

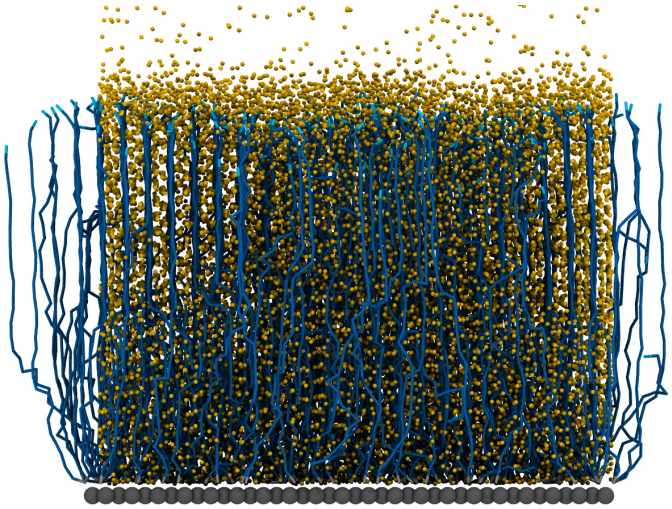

Figure 4.17: *Snapshot of a solvent-induced crystalline polymer brush system with* $\epsilon_{\mathrm{pp}} = 0.6$ *and* $\epsilon_{\mathrm{ps}} = 2.0$.

Octopus micelles

In some cases, when the grafting density of a polymer brush is close to critical and the solvent quality is suddenly reduced, surface structures called octopus micelles can form [41]. These structures are characterised by chains grafted from different locations clumping together in a single globule upon collapse, resulting in nanoscale surface segregation.

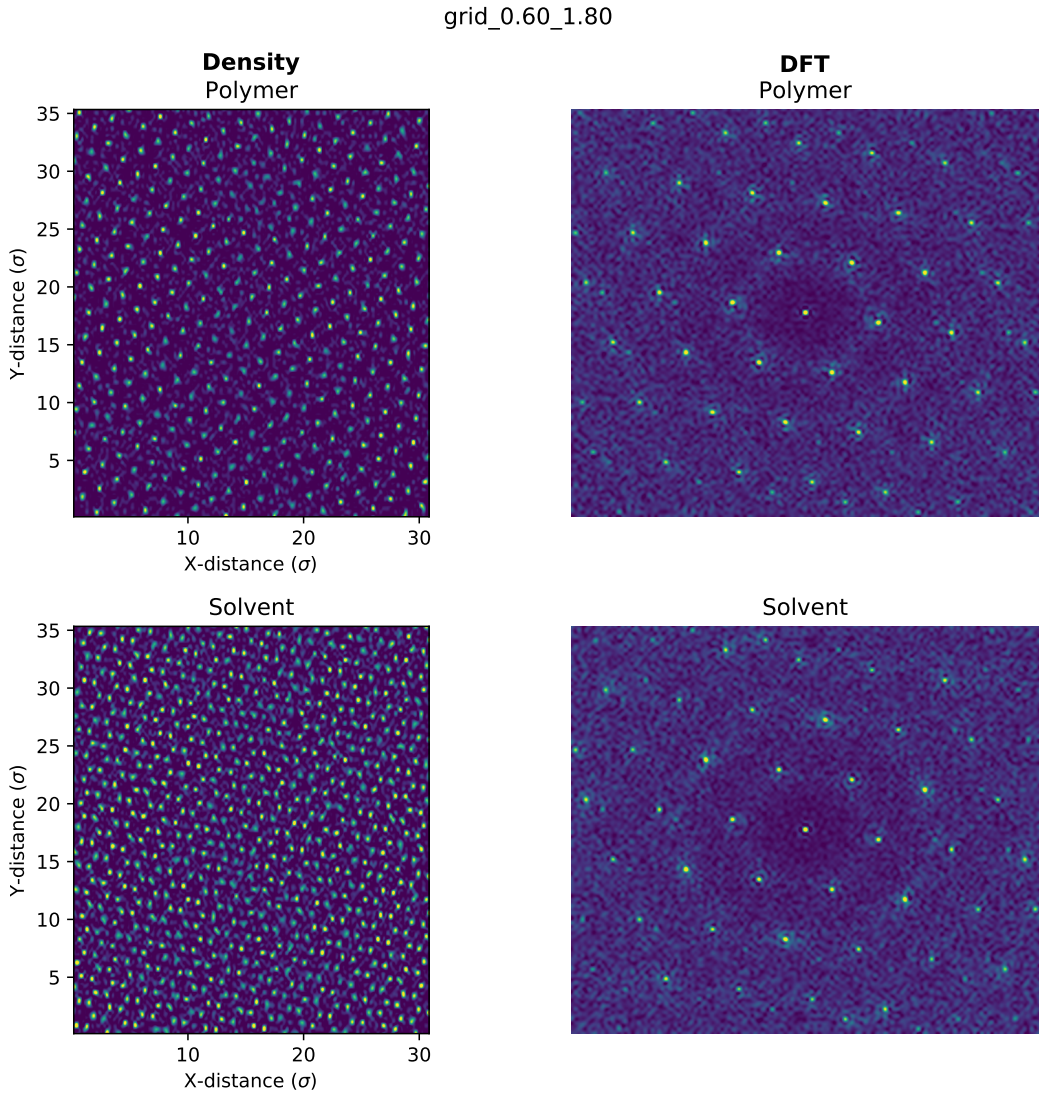

Figure 4.18: *xy polymer and solvent density maps (**left**) and the magnitude of their DFTs (**right**) for a brush system with* $\epsilon_{\mathrm{pp}} = 0.6$ *and* $\epsilon_{\mathrm{ps}} = 1.8$.

Interestingly, octopus micelles are the true free energy minimum (equilibrium), but are only formed when the solvent quality is suddenly reduced, as the entangled chains then collapse together in a collective globule. When the solvent quality drop is gradual, individual chains first get the chance to relax and collapse into individual globules in the intermediate solvent quality state which then prevents further contact with adjacent globules when the solvent quality decreases more (Figure 4.19). Hence, the uniform film of collapsed polymer is actually a metastable, frustrated state which is only formed when solvent quality is slowly decreased. This is unlike most other metastable systems such as glasses, which are formed by suddenly cooling a system so it gets kinetically frustrated before it can reach the equilibrium state. [41]

This behaviour was found to occur with the Kremer-Grest simulations for a grafting density of $0.17\ \sigma^{-2}$, $\epsilon_{\mathrm{pp}} = 1.0$, and $\epsilon_{\mathrm{ps}} = 1.0$ (Figure 4.20). It was no longer found to occur when the grafting density was increased to $0.34\ \sigma^{-2}$.

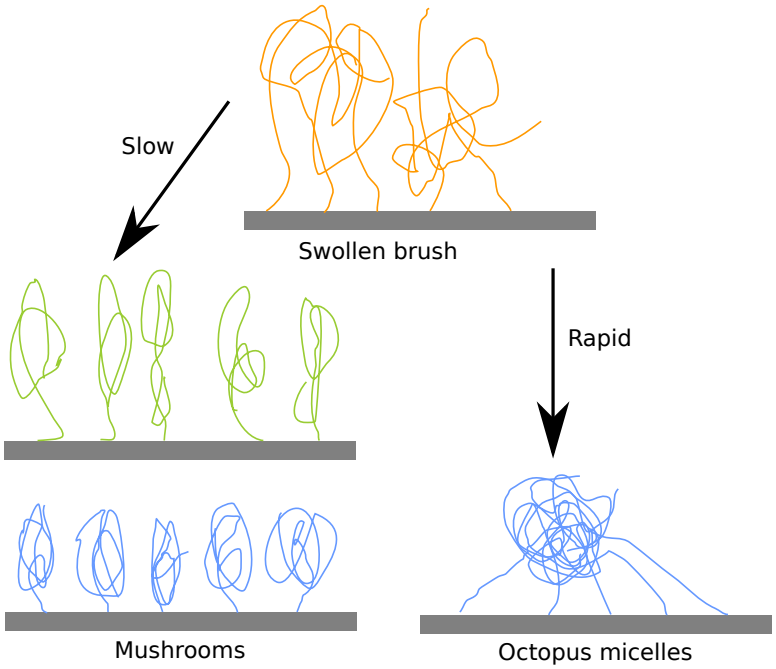

Figure 4.19: *Illustration of the formation of octopus micelles upon decreasing solvent quality.*

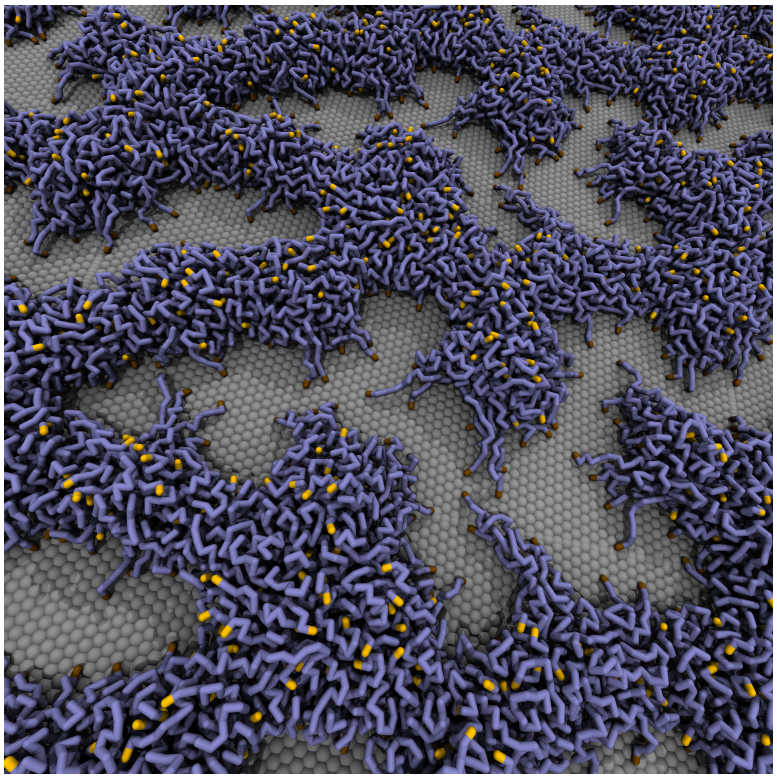

Figure 4.20: *Snapshot of a simulation showcasing octopus micelles.*

4.2.3 MARTINI

Determination of saturation pressure

In Figure 4.21, density and pressure profiles of the coexistence simulations are shown in a similar fashion to Figure 4.4. Compared to the analogous simulations with generic Lennard-Jones fluid for the Kremer-Grest simulations, the pressure profiles here show much more fluctuations in the liquid phase. The fluctuations in the liquid phase are higher than in the vapour phase, which is strange because statistically one would expect the opposite since the number density in the liquid phase is orders of magnitude higher. The reason for this behaviour is as of now unclear.

Nevertheless, the saturation pressure was determined by performing the construction for a range of overall densities leading to the plot shown in Figure 4.22. A negative linear trend is visible, but this is attributable to the effect that at higher overall densities, more of the fluid is in the liquid phase and thus the fluctuations in the pressure are given more weight. The true saturation pressure is probably closer to the upper value (~ 0.185) than to the lower value (~ 0.155). For now, we go with a value of $p_{\mathrm{sat}} \approx 1.75$.

PE

The MARTINI PE brush is in many ways different and more realistic than the Kremer-Grest system, but in other aspects also quite similar. PE coarse-grained using MARTINI is still a fully linear chain of beads, just like Kremer-Grest. MARTINI also still uses the same Lennard-Jones potential between non-bonded particles (although the truncation and shifting is different). Hence, it is not surprising that the behaviour of the MARTINI PE system is not that different.

The MARTINI PE model does differentiate itself in the mechanical behaviour because chain stiffness is more accurately represented. In the Kremer-Grest model, chain stiffness is completely coarse-grained away as the bead size is equal to the polymer's Kuhn unit. The MARTINI PE model features a somewhat smaller bead size and does implement angle and dihedral constraints which give rise to a chain stiffness. However, this does not really influence sorption behaviour in any way.

Density profiles

The PE brush solvated by acetone ($\mathrm{N_a}$ beads) vapour shows a little bit of adsorption at the brush border and near the substrate, but no significant absorption. In terms of a comparison to the Kremer-Grest results, the density profile is similar to a profile with low-to-moderate polymer-solvent and polymer self-affinities, like $\epsilon_{\mathrm{pp}} \approx 0.6$ and $\epsilon_{\mathrm{ps}} \approx 0.6$.

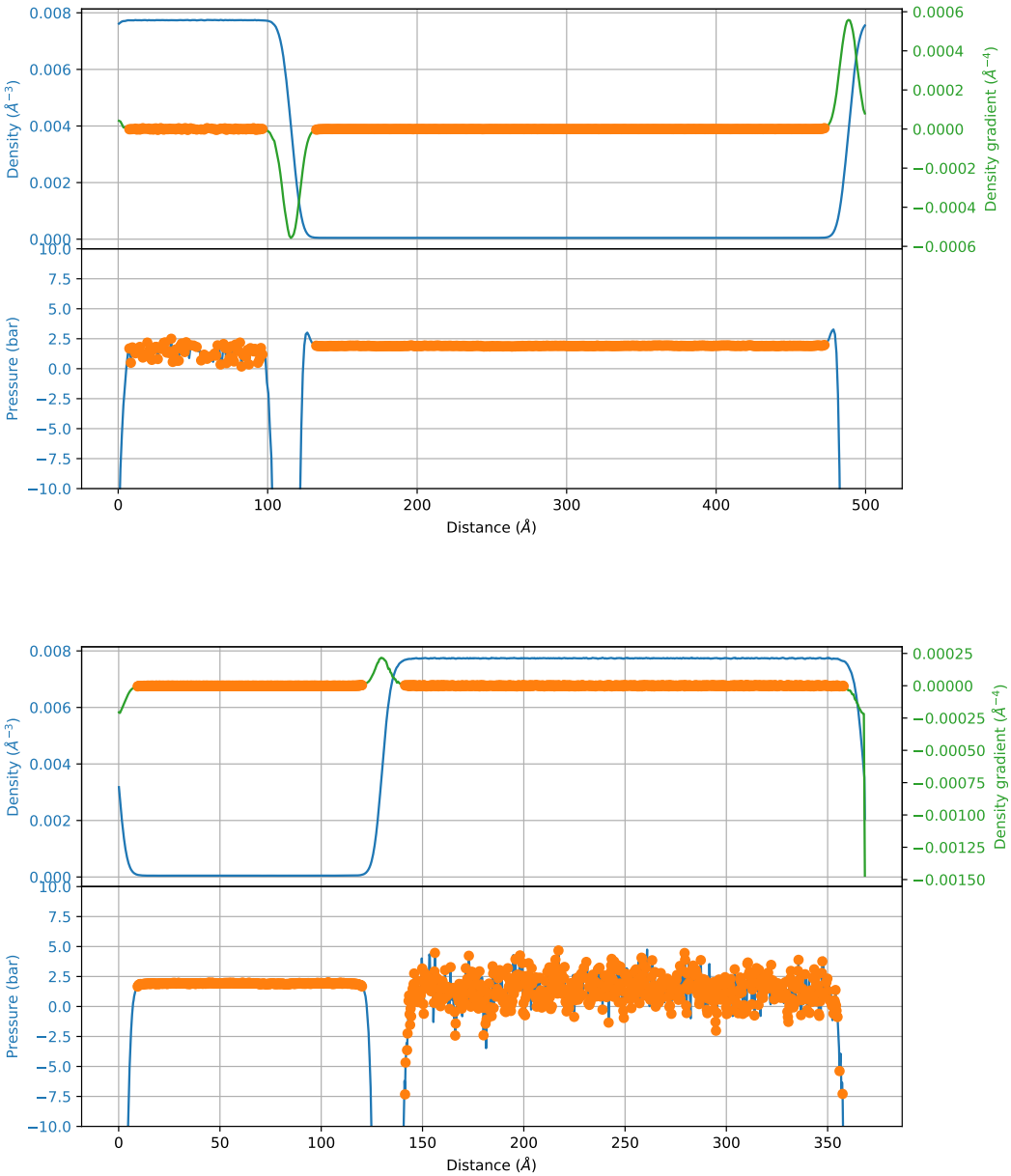

Figure 4.21: *Density (top) and pressure (bottom) profiles for* $\rho = 0.002\,\text{Å}^{-3}$ *(left) and* $\rho = 0.005\,\text{Å}^{-3}$ *(right). The orange points indicate the bulk phases.*

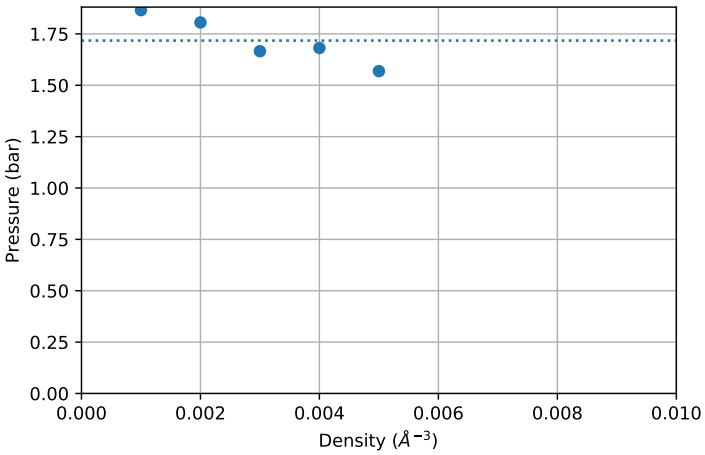

Figure 4.22: *Mean pressure in the bulk of both phases as a function of overall density.*

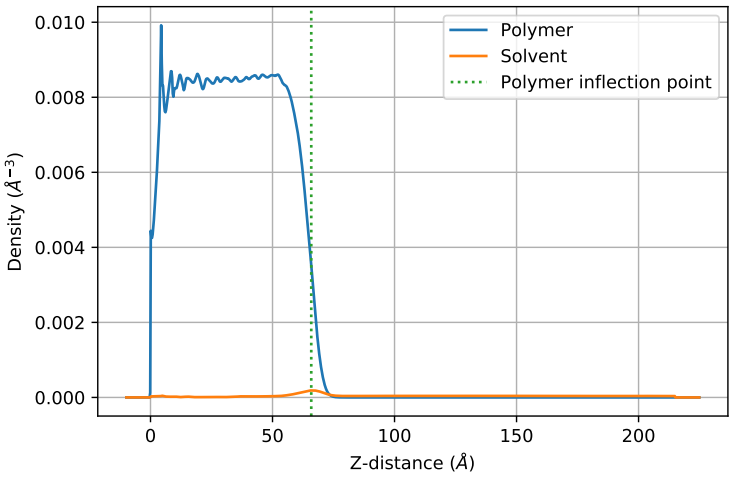

Figure 4.23: *Polymer and solvent density profiles of a MARTINI PE in acetone vapour.*

5 | Conclusion and outlook

The vapour-solvated polymer brush simulations using the Kremer-Grest model show that the sorption behaviour can be predicted based on the polymer self-affinity and the polymer-solvent affinity. The adsorption behaviour was found to be controlled by the polymer-solvent affinity. At low polymer-solvent affinity, there is negligible sorption. The amount of adsorption increases with polymer-solvent affinity, but was found to be virtually independent of polymer self-affinity. Absorption was found to be controlled by the relative affinity (W). As such, absorption occurs when the polymer self-affinity is sufficiently low, and the polymer-solvent affinity sufficiently high.

When varying the solvent pressure, the sorption increases smoothly with the solvent pressure. The swelling was also found to be perfectly linear with absorbed solvent, which proves that no filling of free volume occurs.

At high vapour pressures, very thick adsorption layers are formed. It was demonstrated that accurate reproduction of this phenomenon depends strongly on the polymer-solvent cut-off, but that a cut-off of $r_c = 2.5\ \sigma$ is sufficiently large, since the thickness plateaus at this point.

The Kremer-Grest model is questionable in several regards. For one, in this model the solvent particles are equal in size to the coarse-grained beads (Kuhn units) of the polymer, which is far from realistic for most polymers and solvents. Moreover, the interaction parameters (ϵ_{pp} and ϵ_{ps}) used here are most probably very extreme compared to values that accurately model real-life polymer and solvent combinations. Likewise, the LJ fluid is rather volatile compared to real-life fluids like water or acetone.

The simulation of PE in acetone vapour, modelled by MARTINI, showed no drastic differences. The MARTINI PE model suffered from severe numerical stability issues in LAMMPS stemming from the use of a angle potential with an equilibrium angle of 0 with a dihedral potential. For further use of this model, it might be worthwhile to look into implementing the combined bending-torsion (CBT) potential in LAMMPS, which does not suffer from this problem.

PE in acetone showed very little sorption, as is expected because in real-life acetone is not a particularly good solvent for PE. In follow-up work, it will be interesting to simulate different solvents such as alkanes, which can be modelled by MARTINI C_1 particles, with PE. Similarly, different polymers, like poly(ethyleneglycol) (PEG) or poly(styrene) (PS) will be interesting.

A | Excursions in computer and data science

A.1 Poisson-disk point set generation algorithms

A Poisson-disk point set is a set of points randomly sampled from a uniform distribution, with the constraint that no pair of points is closer than a given distance r [42, 43].

These sets see use as sources of 'blue noise', for example for dithering noise in image processing. However, for us, they are important for application as grafting points on a substrate; after all, when creating a polymer brush system to use as initial conditions for a MD simulation, we want to 'graft' the polymers in a random, uniform fashion to plane, however, just sampling points from a uniform distribution would inevitably yield some points that are unrealistically close to each other. In a real-world system, two grafting points would never be closer than a certain minimum distance, even in grafting-from, because steric hindrance prevents two anchoring molecules to adsorb to the substrate closer than their size.

A.1.1 Naive sifting

It is not trivial to come up with an efficient algorithm for generating Poisson-disk point sets. A simple first candidate algorithm would be to sample n points from an uniform distribution, and then sift through all pairs of points using nested loops to look for 'overlapping' points and move those. The number of pairs is equal to n choose 2:

$$\binom{n}{2} = \frac{n!}{2!(n-2)!} = \frac{(n-1)n}{2} \tag{A.1}$$

hence, keeping the point density (n divided by the domain area) and r constant, this algorithm's time complexity is $O(n^2)$, which is already not ideal. Additionally, when the desired point density is very high compared to r, this algorithm takes very long to converge. In practice, this means this approach is entirely unfeasible if densities higher than approximately half the close packing density (Equation A.2) are desired.

$$\eta_{\mathrm{h}} = \frac{2\sqrt{3}}{3r^2} \tag{A.2}$$

A.1.2 Naive dart throwing

It is more efficient to add points one by one, checking for overlap after sampling each point from an uniform distribution. This algorithm also runs in $O(n^2)$ time (at constant density), but is less computationally expensive than naive sifting. Just like naive sifting, it takes asymptotically longer to converge when densities become higher, because the probability of a new point being accepted approaches 0 as the density approaches the maximal Poisson-disk point set. One drawback to naive dart throwing is that unlike the naive sifting approach, it is not possible to attain densities close to the close packing density (even with unlimited computing time), because existing points cannot be moved. The highest attainable density is the maximal Poisson-disk point set.

Both the naive sifting and the naive dart throwing algorithms require an iteration limit (of the outer loop) in practice because of the prohibitively huge execution times for higher n and point densities. The difference between the two algorithms is that the point set the naive sifting algorithm returns when it is halted by the iteration limit is not a valid Poisson-disk point set because it will contain points that are closer than r, whereas the naive dart throwing algorithm does return a valid Poisson-disk point set in that case, albeit an incomplete one; it will contain less points than requested.

A.1.3 Dart throwing accelerated by a cell list

An even better approach is to first subdivide the domain into square cells, where the cell size is as large as possible while still being totally covered by the 'exclusion circle' of a point anywhere in the cell. This results in a cell size of $\frac{r}{\sqrt{2}}$ (see Figure A.1a). This way, every cell can at most contain one point, so if we know that a cell contains a point, we don't need to try to put another point in it. We can also use this grid to accelerate the search for overlapping points, because we only need to look in the 20 neighbouring cells (see Figure A.1b).

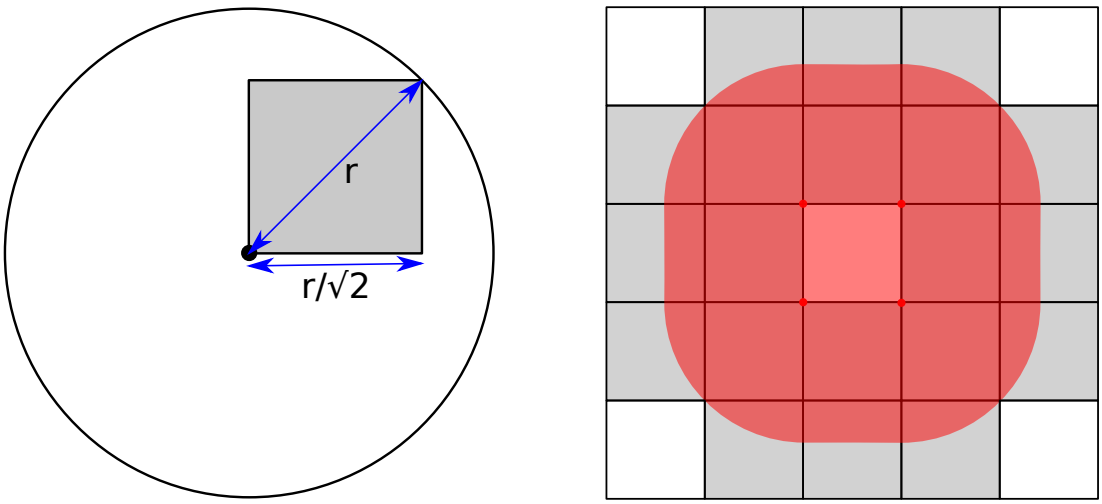

***Figure A.1: a**: $\frac{r}{\sqrt{2}}$ is the maximum size of the cells while they are still totally covered by a point anywhere inside them. **b**: 20 cells surrounding the central cell could contain points that overlap with points in the central cell: a 5x5 grid excluding the central cell and the 4 corner cells.*

This algorithm functions in roughly the following way:

1. Divide the domain into square cells of size $\frac{r}{\sqrt{2}}$ and put them in a (flat) list of active cells.
2. While the number of points is less than the desired number of points:
   (a) Choose an active cell and 'throw a dart' in it: sample a point from an uniform distribution.
   (b) Check for overlap by checking the distance to all points in the 20 neighbouring cells (a 5x5 grid centred around the current cell, excluding the current cell itself and the 4 corner cells) using the cell lookup list (Figure A.1b). If there is no overlap, add the point to the total list and to the cell lookup list, and remove the current cell from the active cells list.

Two data structures are maintained: the active cells list, which is a list containing tuples of cell indices of the cells that do not contain a point. It starts out containing all the cells. When a point is added to a cell, the cell is removed form the active cells list. The cell lookup list is a 2D matrix that starts out empty. When a point is added, its coordinates get added to the respective cell in the list, in order to accelerate searching for overlaps.

The big advantage is that this algorithm appears (empirically) to be $O(n)$, i.e. linear in time with respect to n at constant r and point density. This can be seen from Figure A.2. Note that in the constant size case, the iteration limit is hit around $n \approx 90$, as evidenced by the plateau in the "naive dart throwing" data. Also note that the curves in the constant size case do not appear linear in the log-log plot, which means they run in superpolynomial time (probably exponential time).

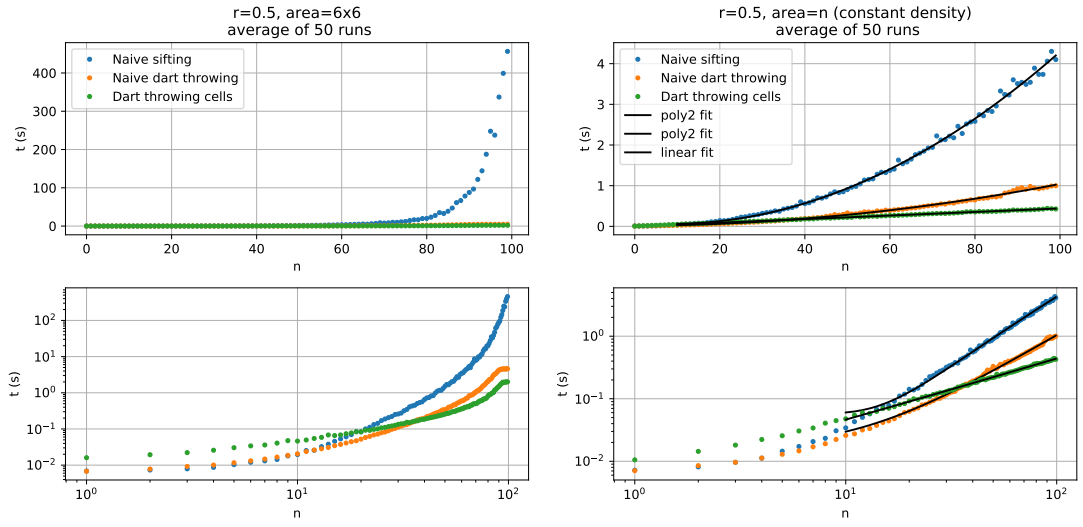

Figure A.2: *Time complexity plots (t vs n) for constant r and domain area (i.e. increasing point density) (**left**) and for constant r and point density (**right**). The bottom plots show the same data, but in log-log form. The constant density cases are shown with polynomial fits (quadratic fits for the naive algorithms, and a linear fit for the "dart throwing cells" algorithm).*

This algorithm can end in one of three ways: either the desired number of points is reached, no active cells are left, or there are active cells left but the point set is maximal and no extra points can be added to the set. In the first case, the result (consisting of a set of point coordinates) is satisfactory. In the second case, the result is a valid point set, but the density is lower than requested. In the third case, the algorithm doesn't halt intrinsically and a iteration limit (of the outer loop) is required to halt the program after some time.

The code can be found at `https://github.com/Compizfox/MD-scripts/blob/master/PoissonDiskGenerator.py`.

A.1.4 More sophisticated algorithms

Cline and Egbert [42] showed the above algorithm can be improved further by recursively subdividing the cells into 4 smaller cells in a quadtree-like fashion every time an overlap test fails in an active cell. This is called "Hierarchical Dart Throwing" and improves the performance further at the cost of some additional implementation complexity.

If *maximal* point sets are required, a very nice algorithm exists by Bridson [44] that is $O(n)$, but it only generates maximal point sets.

A.2 LAMMPS brush generator

For use with the MARTINI simulations, a new brush generator script was developed in Python that is more extensible and thus can handle Kremer-Grest topologies as well as more complex molecular topologies one might encounter in MARTINI models.

The script consists of a generic (*abstract*) class BrushGenerator that defines model-agnostic code for building a polymer brush system and writing the initial data file in the format LAMMPS accepts. It makes use of a Poisson-disk point set generator (implemented in a separate class PoissonDiskGenerator) as described above in A.1.3 to generate coordinates of the grafting layer.

Model-specific classes subclass (inherit from) BrushGenerator, with the most important method being overridden being `_build_bead()`. This method is called iteratively in `build()` and should be implemented according to the polymer model. Its function is to 'build a bead' of the polymer chain and append the coordinates, bonds, angles, and dihedrals (if applicable) to the object's non-final lists.

When `build()` is run, it calls `_build_bead()` for every bead for every chain and converts the non-final lists to Pandas DataFrames afterwards. Then, in `write()`, the DataFrames are read and converted to a LAMMPS data file, together with force field coefficients.

This way, the code that does not depend on the exact polymer model is contained in a single, abstract class which can easily be subclassed to implement a specific polymer model with minimal extra code.

The code can be found at `https://github.com/Compizfox/MD-scripts/tree/master/BrushGenerators`.

B | About LAMMPS performance and scaling

There are several factors impacting the total resultant performance (as usually measured in simulation time per wall time, e.g. $\mathrm{ns\,d^{-1}}$). On 'the bottom' (the lowest level) we have the performance of the hardware (which can, for a certain configuration of hardware, also be influenced by core clock boosting/throttling[1]). The next level is the operating system kernel (usually Linux, in the case of HPC), followed by the MD code itself and how it is compiled. We limit the scope here to LAMMPS. Finally, the specific system one is simulating and the simulation settings (cut-offs, neighbouring intervals/skin sizes, timesteps, etc) have a big impact.

Because every layer in this stack influences the performance in some way, it is non-trivial to make general statements about performance of MD.

B.1 Compilers

The act of translating the source code of a computer program into machine code is called *compilation* and is performed by a *compiler*. Several compilers exist, even for one programming language (such as C++). Some popular ones for the C language family are:

- GCC (GNU Compiler Collection)
- Clang (C language family frontend for LLVM)
- ICC (Intel C++ Compiler)

The first two are free[2], while the Intel compiler is proprietary. GCC is the most used free compiler and was developed as part of the GNU project. Clang is a newer compiler that is built upon the LLVM compiler framework.

The compiler one uses to compile a program can have a pronounced effect on the performance of resulting machine code. Compilers carry out numerous optimisations that can be set using compile-time options and some compilers are better at these optimisations than others. The Intel compiler famously excels at generating very performant machine code for Intel CPUs specifically. However, it is less common on Linux systems and besides rather infamously known for generating sub-optimal code for their competitor's (AMD) processors [45]. Clang has a reputation for the compilation itself being quicker than GCC's, but not generating as well-optimised machine code as GCC does.

[1] Most modern CPUs and GPUs will dynamically 'boost' their clock frequency above their base clock, provided that cooling and power delivery conditions permit it. Generally, if only one CPU core is loaded, the system will only boost that core (*single-core boost*). The maximum single-core boost clock is higher than the *all-core boost* clock.

Similarly, when cooling or power delivery is insufficient a processor can (respectively) *thermal throttle* or *power throttle* (adjust their clock to below the base clock). Most low-power passively-cooled mobile hardware (in laptops) operates in this state under normal conditions, but in desktop/server hardware this is always undesirable.

[2] As in freedom, see: https://www.gnu.org/philosophy/free-sw.en.html

B.2 LAMMPS parallelisation mechanisms

Without applying any clever tricks, the MD algorithms are single-threaded and thus only take advantage of one CPU core. This is not ideal because contemporary CPUs have many cores, and supercomputers can even leverage multiple nodes at once. Luckily, there are several ways to speed up LAMMPS performance by introducing parallelisation.

B.2.1 MPI (spatial domain decomposition)

The most obvious and (in most cases) most effective approach to this is to divide the simulation volume (domain) into smaller subdomains in a process called spatial domain decomposition [22, 35]. Every subdomain is then assigned to a separate CPU core, which runs an independent instance of LAMMPS. This does however inevitably introduce complications: the simulation needs to know what happens just over the interfaces of the subdomains because particles near the interfaces are within the cut-off range of particles that might be on the other side. Particles can also move over the interface, and need to be handed over to the different LAMMPS instance. In other words: there needs to be a means of communication. This is realised by the Message Passing Interface (MPI): a general-purpose protocol for communication between parallel instances of a wide variety of programs. Several implementations of MPI exist, of which OpenMPI is the most popular example.

The communication overhead is proportional to the total interfacial area of the subdomains. Therefore, LAMMPS tries to minimise this interfacial area for a requested number of subdomains. Another consequence of this fact is that at some point the speed-up gained by subdividing the domain is overshadowed by the communication overhead brought along with it, at which point the performance scaling plateaus: at this point it is said that the simulation “does not scale" any more. [35]

Sub-domain balancing

If the simulated system is perfectly homogeneous (no large-scale density variations exist), we don’t have many issues; the workload is evenly distributed over all MPI ranks. However, if this is not the case, and this is often not the case, some subdomains contain many more particles and other subdomains might be almost (or fully!) empty. This hurts our parallelisation efforts because some of the MPI ranks are now not doing much at all.

We can correct this problem by making the division of the domain into subdomains non-uniform, thus increasing the volume of subdomains in sparse regions of the simulation domain, while subdividing dense regions into more, smaller subdomains with the goal of balancing the number of particles (and thereby the approximate workload) over the subdomains.

Inhomogeneities arise for example in vapour-liquid coexistence simulations, where an interface separates the much denser liquid phase from the vapour phase, but also in polymer brush simulations where the bottom of the simulation volume contains grafted polymer, but the top of the box is empty or contains a solvent vapour. In both of the aforementioned examples, the system is homogeneous in x and y; the density variations only exist over z. This simplifies the balancing problem considerably.

Balancing can be performed once (`balance` in LAMMPS) or periodically (`fix balance`), the latter being useful if the density variations change with time (for example, because the polymer brush collapses/swells during the course of the simulation).

B.2.2 OpenMP (threading)

An alternative to spatial domain decomposition is multi-threading of the computations, such as force computations and neighbouring themselves. This is possible in LAMMPS through the `USER-OMP` package, which implements threading through the OpenMP libary. [17]

OpenMP threading can be enabled in conjunction with spatial domain decomposition (MPI), in which case every MPI rank uses a number of OpenMP threads.

It is hard to make general statements about the optimal number of OpenMP threads as this is very sensitive to the specific system that is simulated, but in general OpenMP threading is most effective when a few threads are used. MPI parallelisation is often more effective, but in cases where the bandwidth between processors running different MPI tasks is very high, or in the case where the scaling limit of MPI parallelisation is reached, it can be beneficial to reach out to OpenMP. [17]

B.3 Hardware: building a Threadripper-powered MD machine

A new PC (Figure B.1) specifically for running MD simulations was designed and built on basis of the new AMD Ryzen Threadripper 3970X. This is a 32-core CPU with SMT and a base clock of 3.7 GHz and a maximum boost clock of 4.5 GHz (single-core). An overview of the components of the PC is listed in Table B.1

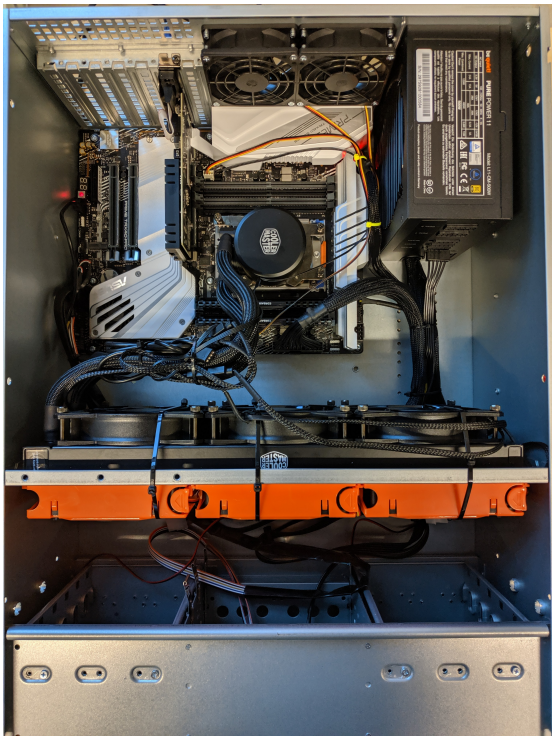

Figure B.1: *Picture of the new MD PC.*

Table B.1: *PC components*

CPU	AMD Ryzen Threadripper 3970X
Mother-board	Asus Prime TRX40-Pro
RAM	Corsair Vengeance LPX 16 GB (2 x 8 GB) DDR4-3200
Storage	Samsung 970 Evo Plus 500 GB
Graphics	Zotac GeForce GT 710
PSU	be quiet! Pure Power 11 CM 500 W
CPU cooler	Cooler Master MasterLiquid ML360 RGB TR4 Edition

The CPU is cooled by a 3x $120\,\mathrm{mm}$ fan all-in-one liquid cooler. The graphics card is purely for video output, as the Threadripper does not have integrated graphics. At this point, it was decided to not yet invest in high-performance GPUs. Although GPUs are promising for molecular dynamics, at this moment we have not yet benchmarked our simulations in LAMMPS on GPUs so the speed-up is unclear. There is the possibility to add GPUs later on, however.

The new MD PC is intended to replace the four old i7-4790-based PCs used for MD currently. The benchmark results are compared below in B.5.2.

B.4 Benchmark method

In order to measure the performance of LAMMPS in a reproducible way, a simple Lennard-Jones benchmark simulation was set-up consisting of 100000 MARTINI $\mathrm{N_a}$ particles interacting by LAMMPS' `lj/gromacs` potential at a density of $1 \times 10^{-5}\,\mathrm{Å}^{-3}$. At this density, the Lennard-Jones fluid exists in the vapour phase, so any phase separation leading to inhomogeneity and thereby spatial computational imbalance is avoided. The neighbour skin distance is $2\,\mathrm{Å}$ and the timestep is $10\,\mathrm{fs}$. The system is first minimised using the conjugate gradient method (`min_style cg`) and dynamics are then run for 10000 timesteps. The simulation performance (in $\mathrm{ns\,d^{-1}}$) of the dynamics run is measured.

B.5 Benchmark results and discussion

B.5.1 Compiler effect

When comparing GCC 8.3.0 with Clang 8.0.1, both with the compiler flags `-O3 -DNDEBUG -march=native` on the Threadripper system (see B.3), it was found that LAMMPS compiled by Clang was consistently 15% faster than LAMMPS compiled by GCC (Figure B.2). This is a non-negligible effect and an interesting one considering that Clang usually produces less-optimised code than GCC does. However, this result agrees with that of a comparison test by Phoronix from 2016 [46]. Unfortunately, we were not able to find any more recent third-party benchmark results comparing the effect of compilers on the performance of LAMMPS.

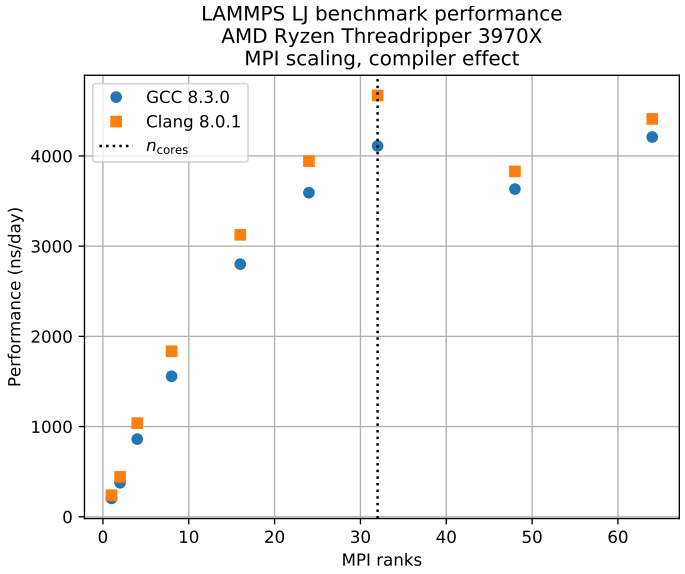

Figure B.2: *MPI performance scaling of the LJ benchmark on the Threadripper system compared for Clang and GCC compilers. The core count (32) is denoted by the vertical line.*

B.5.2 MPI scaling

In Figure B.3, the performance scaling across several systems is compared: the old simulation machine featuring an Intel i7 4790 (4 cores with SMT), the new AMD Threadripper 3970X system (32 cores with SMT) and Cartesius (Dutch National Supercomputer) featuring two Intel E5-2690 v3 CPUs (12 cores with SMT) per node. From both the i7 4790 and the TR 3970X it is very clear that SMT does not provide any additional benefit here; the performance even suffers when more MPI ranks than physical cores are used. This particular benchmark system scales rather linearly with the number of MPI ranks, up until the physical core count of the system is reached. The 48-rank Cartesius result was obtained by employing two nodes[3] and shows that the parallelisation limit of the system is still not reached, because the performance still scales almost linearly at that point.

Smaller systems will show diminishing results of MPI scaling at a much earlier point, as they will contain much less atoms per subdomain and/or have to communicate particle properties to other subdomains much more frequently, which means that the extra communication overhead incurred does not weigh up against the performance gain of parallelisation.

B.5.3 Single-core performance

In Figure B.4, the performance of the benchmark with just one MPI rank is shown for the entire range of systems and compilers addressed above, as well as a new result with the `OMP` versions of the LAMMPS styles enabled with just one thread. These are faster than unaccelerated styles, even with one thread per MPI rank, because they contain several optimisations that reduce overhead.

[3] Cartesius, being a supercomputer, features high-bandwidth interconnects that make it possible to span one simulation over multiple nodes using MPI.

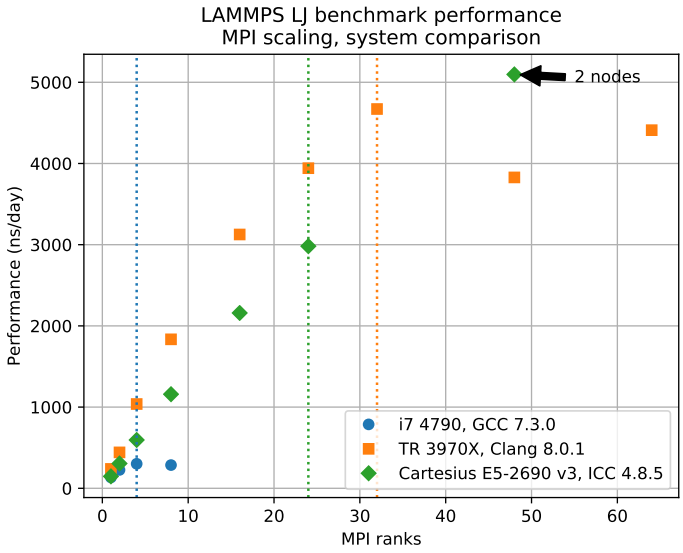

Figure B.3: *MPI performance scaling of the LJ benchmark on different systems. The core count of each respective system is denoted by the vertical lines. The 48-rank Cartesius result was obtained using 2 nodes.*

B.5.4 OpenMP scaling

Figure B.5 shows the performance scaling of the LJ benchmark with the `OMP` versions of the LAMMPS styles enabled with just one MPI rank as a function of the number of OpenMP threads. For all simulations is 2x OpenMP faster than 1x, but the effect is disastrous on the Threadripper system with more threads than that. On Cartesius however, OpenMP still scales up to 16 threads, although not nearly as efficiently as MPI would.

It is well-known that OpenMP is less effective for parallelisation of LAMMPS than MPI in almost all cases, but OpenMP threading should still be beneficial when the number of cores on the system is not exhausted (which is certainly not the case for 1x MPI) [17]. It is unclear why using more than 2 OpenMP threads on the Threadripper system is so disadvantageous for performance. As ICC was used on Cartesius, but only GCC and Clang on the Threadripper, it is possible that this difference in behaviour is caused by the compiler. This will have to be investigated in the future.

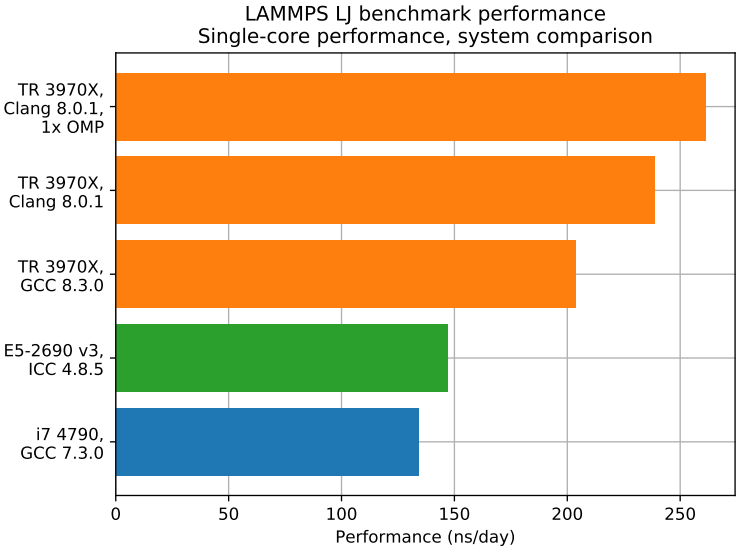

Figure B.4: *Single-core (1 MPI rank) performance on different systems and compilers.*

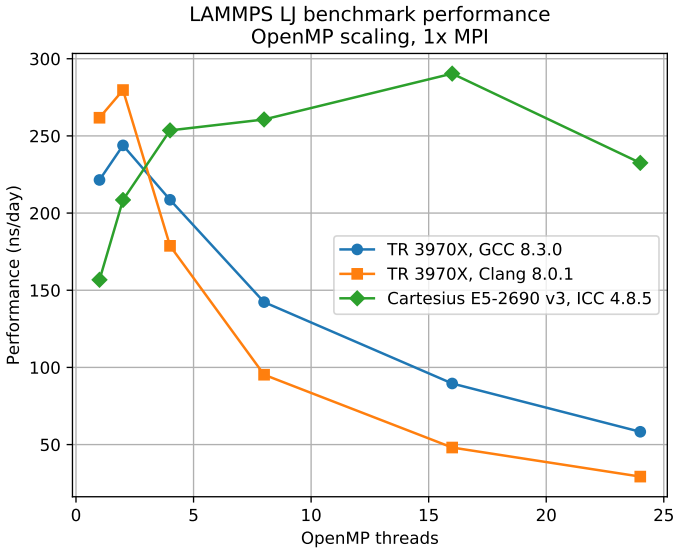

Figure B.5: *OpenMP perfomance scaling of the LJ benchmark on different systems and compilers (1 MPI rank). The lines connecting markers are meant to guide the eye.*

Bibliography

[1] E. P. K. Currie, W. Norde, and M. A. Cohen Stuart. "Tethered polymer chains: surface chemistry and their impact on colloidal and surface properties". In: *Advances in Colloid and Interface Science* 100-102 (2003), pp. 205–265. DOI: 10.1016/S0001-8686(02)00061-1.

[2] W.-L. Chen, R. Cordero, H. Tran, and C. K. Ober. "50th Anniversary Perspective: Polymer Brushes: Novel Surfaces for Future Materials". In: *Macromolecules* 50.11 (2017), pp. 4089–4113. DOI: 10.1021/acs.macromol.7b00450.

[3] S. Tas et al. "Chain End-Functionalized Polymer Brushes with Switchable Fluorescence Response". In: *Macromolecular Chemistry and Physics* 220.5 (2019), p. 1800537. DOI: 10.1002/macp.201800537.

[4] H. C. McCaig, E. Myers, N. S. Lewis, and M. L. Roukes. "Vapor Sensing Characteristics of Nanoelectromechanical Chemical Sensors Functionalized Using Surface-Initiated Polymerization". In: *Nano Letters* 14.7 (2014), pp. 3728–3732. DOI: 10.1021/nl500475b.

[5] Y. Higaki, M. Kobayashi, D. Murakami, and A. Takahara. "Anti-fouling behavior of polymer brush immobilized surfaces". In: *Polymer Journal* 48.4 (2016), pp. 325–331. DOI: 10.1038/pj.2015.137.

[6] J. Klein, E. Kumacheva, D. Mahalu, D. Perahia, and L. J. Fetters. "Reduction of frictional forces between solid surfaces bearing polymer brushes". In: *Nature* 370.6491 (1994), pp. 634–636. DOI: 10.1038/370634a0.

[7] S. de Beer, E. Kutnyanszky, P. M. Schön, G. J. Vancso, and M. H. Müser. "Solvent-induced immiscibility of polymer brushes eliminates dissipation channels". In: *Nature Communications* 5.1 (2014), pp. 1–6. DOI: 10.1038/ncomms4781.

[8] Y. Yu, B. D. Kieviet, E. Kutnyanszky, G. J. Vancso, and S. de Beer. "Cosolvency-Induced Switching of the Adhesion between Poly(methyl methacrylate) Brushes". In: *ACS Macro Letters* 4.1 (2015), pp. 75–79. DOI: 10.1021/mz500775w.

[9] C. J. Galvin and J. Genzer. "Swelling of Hydrophilic Polymer Brushes by Water and Alcohol Vapors". In: *Macromolecules* 49.11 (2016), pp. 4316–4329. DOI: 10.1021/acs.macromol.6b00111.

[10] C. J. Galvin, M. D. Dimitriou, S. K. Satija, and J. Genzer. "Swelling of Polyelectrolyte and Polyzwitterion Brushes by Humid Vapors". In: *Journal of the American Chemical Society* 136.36 (2014), pp. 12737–12745. DOI: 10.1021/ja5065334.

[11] L. Sun et al. "Scaling Behavior and Segment Concentration Profile of Densely Grafted Polymer Brushes Swollen in Vapor".
In: *Langmuir* 32.22 (2016), pp. 5623–5628. DOI: 10.1021/acs.langmuir.6b00845.

[12] J. H. W. H. Nijkamp.
"Revealing solvent partitioning in vapour-hydrated polymer brushes".
Enschede: Universiteit Twente, 2019.

[13] R. A. L. Jones. *Soft Condensed Matter*. OUP Oxford, 2002. 212 pp.
ISBN: 978-0-19-850589-1.

[14] C. Holm. *Simulation Methods in Physics*.
Institute for Computational Physics, University of Stuttgart.
URL: https://www2.icp.uni-stuttgart.de/~icp/mediawiki/images/5/54/Skript_sim_methods_I.pdf (visited on 2019-12-13).

[15] D. Frenkel and B. Smit.
Understanding Molecular Simulation: From Algorithms to Applications.
Computational science. Elsevier Science, 2001. ISBN: 978-0-08-051998-2.

[16] M. Tuckerman, B. J. Berne, and G. J. Martyna.
"Reversible multiple time scale molecular dynamics".
In: *The Journal of Chemical Physics* 97.3 (1992), pp. 1990–2001.
DOI: 10.1063/1.463137.

[17] Sandia National Labs. *LAMMPS Documentation*.
URL: https://lammps.sandia.gov/doc/Manual.html (visited on 2020-01-17).

[18] D. Schroeder. *An Introduction to Thermal Physics*. Addison Wesley, 1999.
ISBN: 978-0-201-38027-9.

[19] K. Kremer and G. S. Grest. "Dynamics of entangled linear polymer melts: A molecular-dynamics simulation".
In: *The Journal of Chemical Physics* 92.8 (1990), pp. 5057–5086.
DOI: 10.1063/1.458541.

[20] S. Toxvaerd and J. C. Dyre.
"Communication: Shifted forces in molecular dynamics".
In: *The Journal of Chemical Physics* 134.8 (2011), p. 081102.
DOI: 10.1063/1.3558787.

[21] B. Hafskjold et al. *Thermodynamic properties of the 3D Lennard-Jones/spline model*. 2019. arXiv: 1907.07039. (Visited on 2019-08-28).

[22] V. Eijkhout. *Introduction to High Performance Scientific Computing*. lulu.com, 2011.
ISBN: 978-1-257-99254-6. DOI: 10.5281/zenodo.49897.

[23] C. Abrams. *Molecular Simulations*.
Department of Chemical Engineering, Drexel University, Philadelphia. 2013.
URL: http://www.pages.drexel.edu/~cfa22/msim/msim.html (visited on 2020-01-16).

[24] V. Rühle. *Berendsen and Nose-Hoover thermostats*. 2007. URL: https://www2.mpip-mainz.mpg.de/~andrienk/journal_club/thermostats.pdf (visited on 2019-10-23).

[25] S. C. Harvey, R. K.-Z. Tan, and T. E. Cheatham. "The flying ice cube: Velocity rescaling in molecular dynamics leads to violation of energy equipartition". In: *Journal of Computational Chemistry* 19.7 (1998), pp. 726–740. DOI: 10.1002/(SICI)1096-987X(199805)19:7<726::AID-JCC4>3.0.CO;2-S.

[26] E. Braun, S. M. Moosavi, and B. Smit. "Anomalous Effects of Velocity Rescaling Algorithms: The Flying Ice Cube Effect Revisited". In: *Journal of Chemical Theory and Computation* 14.10 (2018), pp. 5262–5272. DOI: 10.1021/acs.jctc.8b00446.

[27] E. Carlon, M. Laleman, and S. Nomidis. "Molecular Dynamics Simulations". 2016. URL: http://itf.fys.kuleuven.be/~enrico/Teaching/molecular_dynamics_2015.pdf (visited on 2019-12-27).

[28] B. Hess. "Stochastic Concepts in Molecular Simulation". PhD thesis. Groningen: Rijksuniversiteit Groningen, 2002. URL: https://www.rug.nl/research/portal/files/3062080/thesis.pdf (visited on 2020-01-16).

[29] L. Sjögren. "Brownian Motion: Langevin Equation". In: *Stochastic processes*. URL: http://physics.gu.se/~frtbm/joomla/media/mydocs/LennartSjogren/kap6.pdf (visited on 2020-01-17).

[30] P. S. Doyle and P. T. Underhill. "Brownian Dynamics Simulations of Polymers and Soft Matter". In: *Handbook of Materials Modeling: Methods*. Ed. by S. Yip. Dordrecht: Springer, 2005, pp. 2619–2630. ISBN: 978-1-4020-3286-8. DOI: 10.1007/978-1-4020-3286-8_140.

[31] Y. Li, B. C. Abberton, M. Kröger, and W. K. Liu. "Challenges in Multiscale Modeling of Polymer Dynamics". In: *Polymers* 5.2 (2013), pp. 751–832. DOI: 10.3390/polym5020751.

[32] S. J. Marrink, H. J. Risselada, S. Yefimov, D. P. Tieleman, and A. H. de Vries. "The MARTINI Force Field: Coarse Grained Model for Biomolecular Simulations". In: *The Journal of Physical Chemistry B* 111.27 (2007), pp. 7812–7824. DOI: 10.1021/jp071097f.

[33] B. M. H. Bruininks, P. C. T. Souza, and S. J. Marrink. "A Practical View of the Martini Force Field". In: *Biomolecular Simulations: Methods and Protocols*. Vol. 2022. Methods in Molecular Biology. Humana Press, 2019, p. 27. ISBN: 978-1-4939-9607-0. DOI: 10.1007/978-1-4939-9608-7.

[34] X. Periole and S. J. Marrink. "The Martini Coarse-Grained Force Field". In: *Biomolecular Simulations*. Ed. by L. Monticelli and E. Salonen. Vol. 924. Totowa, NJ: Humana Press, 2013, pp. 533–565. ISBN: 978-1-62703-016-8 978-1-62703-017-5. DOI: 10.1007/978-1-62703-017-5_20.

[35] S. Plimpton. "Fast Parallel Algorithms for Short-Range Molecular Dynamics". In: *Journal of Computational Physics* 117.1 (1995), pp. 1–19. DOI: 10.1006/jcph.1995.1039.

[36] B. Smit. "Phase diagrams of Lennard-Jones fluids".
In: *The Journal of Chemical Physics* 96.11 (1992), pp. 8639–8640.
DOI: 10.1063/1.462271.

[37] E. Panizon, D. Bochicchio, L. Monticelli, and G. Rossi.
"MARTINI Coarse-Grained Models of Polyethylene and Polypropylene".
In: *The Journal of Physical Chemistry B* 119.25 (2015), pp. 8209–8216.
DOI: 10.1021/acs.jpcb.5b03611.

[38] H. Mo, G. Evmenenko, and P. Dutta.
"Ordering of liquid squalane near a solid surface".
In: *Chemical Physics Letters* 415.1 (2005), pp. 106–109.
DOI: 10.1016/j.cplett.2005.08.142.

[39] I. Snook and W. van Megen. "Structure of dense liquids at solid interfaces".
In: *The Journal of Chemical Physics* 70.6 (1979), pp. 3099–3105.
DOI: 10.1063/1.437798.

[40] S. de Beer, W. K. den Otter, D. van den Ende, W. J. Briels, and F. Mugele.
"Can Confinement-Induced Variations in the Viscous Dissipation be Measured?"
In: *Tribology Letters* 48.1 (2012), pp. 1–9. DOI: 10.1007/s11249-011-9905-4.

[41] D. R. Williams. "Grafted polymers in bad solvents: octopus surface micelles".
In: *Journal de Physique II* 3.9 (1993), pp. 1313–1318. DOI: 10.1051/jp2:1993202.

[42] K. B. White, D. Cline, and P. K. Egbert.
"Poisson Disk Point Sets by Hierarchical Dart Throwing".
In: *2007 IEEE Symposium on Interactive Ray Tracing*.
2007 IEEE Symposium on Interactive Ray Tracing. 2007, pp. 129–132.
DOI: 10.1109/RT.2007.4342600.

[43] T. R. Jones and D. R. Karger. "Linear-Time Poisson-Disk Patterns".
In: *Journal of Graphics, GPU, and Game Tools* 15.3 (2011), pp. 177–182.
DOI: 10.1080/2151237X.2011.617173.

[44] R. Bridson. "Fast Poisson disk sampling in arbitrary dimensions". In:
ACM SIGGRAPH 2007 sketches. ACM Press, 2007, 22–es.
DOI: 10.1145/1278780.1278807.

[45] Intel. *Optimization Notice*.
URL: https://software.intel.com/en-us/articles/optimization-notice
(visited on 2020-01-22).

[46] *LLVM Clang 3.9 Mostly Trails GCC In Compiler Performance*. Phoronix.
URL: https://www.phoronix.com/scan.php?page=article&item=llvm-clang-39&num=2 (visited on 2020-01-13).