Polymer Dynamics In Nanocomposites

Michael Stringer
Registration Number: 100195729

Project Partner, Glenn Jones.
Project Supervisor, Prof. Nigel Clarke.

The University of Sheffield

Abstract

Polymers diffusing within polymer-nanoparticle melts that vary from pure melts with no nanoparticles, volume fraction $\Phi_{NP} = 0$, up to high nanoparticle volume fractions of $\Phi_{NP} = 0.5$ have previously been experimentally analysed to determine their diffusive properties. Inter-Nanoparticle distance (ID), polymer Radius of gyration (R_g) and the normalised diffusion coefficient of polymers (D/D_0) have been found to follow a universal curve of D/D_0 against $ID/2R_g$, showing that confinement of polymers leads to a measurable decrease in polymer diffusion.

Polymer melts are computationally simulated with a three dimensional Monte-Carlo random walk using a coarse bead reptation polymer model treatment. The normalised polymer diffusion coefficients of various polymer length and melt densities are similarly studied after collecting the centre of mass movement of the emulated polymers. Several algorithms are used to produce different nanoparticle distributions for 896 polymers to diffuse though, none of which closely follow the observed experimental master curve within their statistical error. Low Φ_{NP} simulations with the pseudo-random placement algorithm observed the most significant decrease in diffusivity, with a minima for the longest 100 monomer polymers (unit length b=1) of $D/D_0 = 0.983 \pm 0.002$ for $\Phi_{NP} = 0.008$. Higher Φ_{NP} simulations have diffusivity increasing away from the universal curve. The increase in diffusivity is possibly a failure in the coarse reptation model due to an increase in the frequency of polymer kinks, which play a key role in polymer mobility.

PHY480 Research Project in Physics & Astronomy

Spring 2014

Contents

1 A polymer in a polymer melt

A polymer is a chain of repeated monomers connected by covalent bonds, common variants of polymers have backbones containing carbon or oxygen.[1] Previous experimental procedure in studying polymer diffusion utilises tracer polymers of deuterated polystyrene (dPS) to analyse the diffusivity of polystyrene (PS) polymers in a matrix of nanoparticles; phenyl-capped silicon particles (SiO_2).[2] The spatial distribution of a polymer at any time is referred to as a conformation; polymers will reach an entropically favourable conformation as they diffuse.[3]

1.1 Experimental method

The diameter of phenyl capped silica is found to be 28nm after analysis by dynamic light scattering (28.7 nm, σ = 0.147), small angle x-ray scattering (28.6 nm, σ = 0.115) and transmission electron microscopy (TEM, 26.3 nm, σ = 0.159),[2] which is comparable to the entangled length of 200 repeated monomers of Styrene $C_6H_5CH{=}CH_2 \sim 1.8$nm (at 140°c),[1]. This similarity leads to appropriate simulation approximation that treats the size of the base unit of monomer chain length, which will be referred to as b, as the same order of size as the nanoparticles.

1.2 Experimental diffusion

Diffusivity D has dimensions $m^2 s^{-1}$ and can be determined from the appropriate solution to Ficks second law [4, 5] (equation 1) provided the volume fraction of dPs is known. erf, t, and h are the error function, annealing time (seconds), and initial dPS thickness respectively (metres).[2] Rutherford backscattering spectrometry (RBS) and ERD are used to obtain the dPs volume fraction profile $\phi(x)$, whereby the energy and spectrum of scattered ions provide the depth and distribution of the dPS.[2] The diffusivity is then calculated for a polymer of known molecular weight (given in $gmol^{-1}$) and tracer layer thickness h.

$$\phi(x) = \frac{1}{2}\left[erf\left(\frac{h-x}{4Dt}\right) + erf\left(\frac{h+x}{4Dt}\right)\right] \quad [2, 6] \qquad (1)$$

The nanoparticle diffusion coefficient may also be determined. The collaboration use the Stoke-Einstein equation 2 where η is viscosity ($gm^{-1}s^{-1}$) and r nanoparticle radius, to find a nanoparticle diffusivity at 170 ° of 4.0 x 10^{-17} cm^2 s^{-1}.[2] This proves the silica nanoparticles are stable compared to the much more mobile dPS polymers, dPS diffusivity typically ranges from 10^{-12} to 10^{-15} cm^2 s^{-1}[2, 1]. This reflects in the production of a computational model, making it a valid assumption to simulate nanoparticles as unmoving.

$$D = \frac{k_B T}{6\pi\eta r} \quad [7, 8] \qquad (2)$$

Elastic recoil detection (ERD) is used to determine the volume fraction of the nanoparticles, Φ_{NP}. 3 MeV Ion bean analysis bombards the polymer melt with He^{2+} recording the recoil spectrum of hydrogen and deuterium.[2] Density is obtained from the solution to one dimensional Ficks law, equation 3; where J is the diffusion flux mol m^{-2} s^{-1}. In addition TEM provides a cross sectional image that confirms the evenly distributed nature of the produced nanocomposite. This distribution is similar in nature to a repetitive distribution of nanoparticles over a small volume; this provides excellent grounding for the repetitive nature of the nanoparticle algorithms found in section 6.4.

$$J = -D\frac{\partial \Phi}{\partial x} \quad [5, 3] \qquad (3)$$

1.3 Recent experimental results

The experimental work on nanocomposite diffusivity published in macromolecules determined that the normalised diffusivity (D/D_0) where D_0 is the polymer diffusivity in a free $\Phi_{NP} = 0$ polymer melt and D is recorded for volume densities ranging from $\Phi_{NP} = 0.05$ to $\Phi_{NP} = 0.5$.

As shown in figure 1, these diffusivities follow a universal relationship when plotted against $ID/2R_g$,[2, 6] where ID (metres) is the inter-nanoparticle distance obtained from RBS and TEM analysis, where Φ_{NP} is determined and used in equation 4, d is the nanoparticle diameter and radius of gyration (metres) is a conformational identity of polymers discussed in section 3.

The extreme effect of volume fractions on polymer melt conformation is demonstrated by the findings; $ID/2R_g \sim 10$ for $\Phi_{NP} = 0.05$ and $ID/2R_g \sim 2$ for $\Phi_{NP} = 0.5$. [2]

$$ID = d\left[\left(\frac{2}{\pi\Phi_{NP}}\right)^{\frac{1}{3}} - 1\right] \quad [2, 6] \qquad (4)$$

The radius of gyration is not observed to change with an increase of Φ_{NP}, such effects have only been observed with experiments involving carbon nanotubes,[9, 10] where the radius of gyration of polymers has been increased as the polymers are stretched between bottlenecks. The computation model is constructed to test if various nanoparticles placement algorithms also follow this universal relationship.

1.4 Motivational aspects

Understanding how polymer diffusion varies in the presence of entropic barriers induced by nanoparticles is applicable to many biological processes such as the folding of molecules within cell cytoplasm or the delivery of virus DNA into a host nucleus. Self-assembly, particularly that of nanoscale constructs and electronics; is an emergent field of research with great potential.[11]

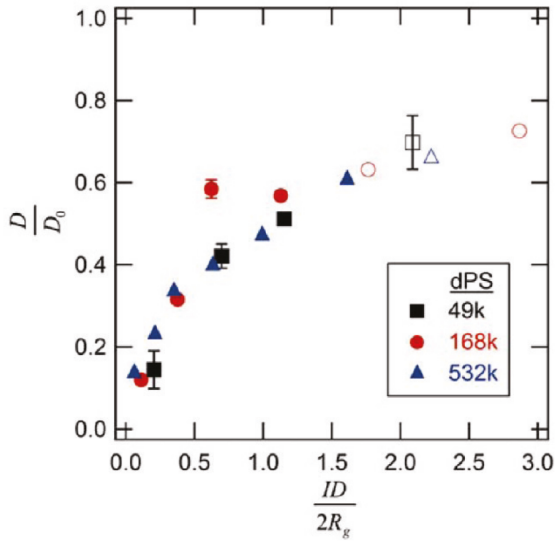


Figure 1: From Macromolecular [2], Experimentally obtained universal scaling law between $ID/2R_g$ and D/D_0. ID is inter-nanoparticle distance (nm), R_g is the radius of gyration (nm), D_0 is the pure polymer diffusion coefficient and D is the polymer melt diffusion coefficient that is observed to decrease with closer spaced nanoparticles.

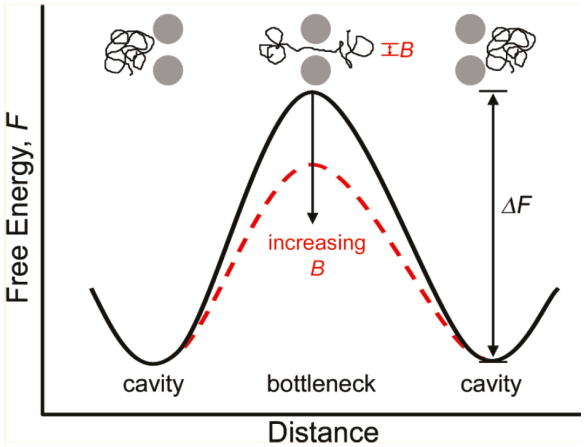


Figure 2: From Macromolecular [2], Free energy F of polymers between nanoparticle induced bottlenecks, size B.

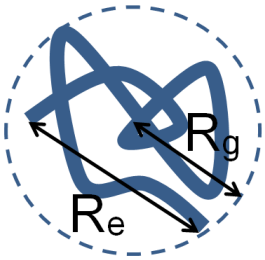


Figure 3: Pictorial representation of polymer end to end vector $\mathbf{R}_e$, and radius of gyration, R_g.

Figure 2 from,[2] demonstrates how the free energy F of a system might change around the cavities and bottlenecks (of size B) that the nanoparticles create. A polymer in a bottleneck will diffuse slower and has the potential to be stretched out with an increase in its radius of gyration. Such an effect might be observable in a simulation. This theory is known as the Entropic Barrier Model (EBM).[12]

2 Polymer dynamics

2.1 Variables of interest

Two key variables are used to follow the diffusive behaviour of polymers over time; these variables keep track of the conformation of the polymer.

Consider a polymer initially extended out in a pure polymer melt, after a period of time diffusion will cause this polymer to naturally reach an equilibrium state similar to that in figure 3.[4] R_e is the end to end vector of the polymer and R_g is the radius of gyration, a property which describes the conformation of the polymer. [3, 4, 8]

Equation 5 defines the root mean squared (RMS) average end to end vector, and RMS average square end to end vector, of polymer length Nb. N is the number of monomer links in a polymer and b monomer link length.

$$< R_e >= 0, \qquad < R_e^2 >= Nb^2 \quad \text{[3, 4, 8, 13]} \tag{5}$$

Equation 6 gives the radius of gyration of a fully freely diffusing polymer in equilibrium.

$$R_g^2 = \frac{Nb^2}{6} \quad \text{[3, 4, 8]} \tag{6}$$

2.1.1 Monte Carlo time step

Within a computational model time is tracked by Monte Carlo time steps. A Monte Carlo method simply refers the repetitive nature of simulating a random walk with pseudo-random numbers and it is convenient for a time step of 1 repetition of code, 1 $T^{(MC)}$,to be equal to the number of polymer beads N. [13] This convenience is a result of the direct relationship between the polymer length and how the polymer position and conformation develop over time as discussed in the following section.

2.2 The Reptation model

The reptation polymer diffusion model is an expansion on the Rouse model,[3] where the polymers adjacent monomers are assumed to be a series of beads connected by springs that follow ideal bead elasticity unaffected by inertia, as shown in figure 4.[3] The only case considered here is for

Figure 4: Monomers in Rouse model, connected by springs.

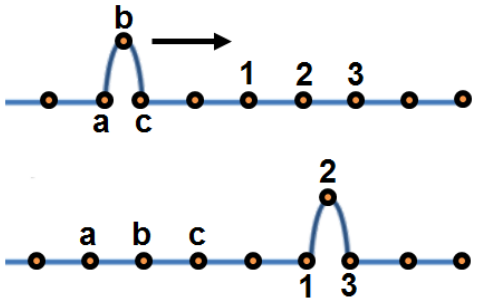


Figure 5: Hairpins of 2 monomers beads a,b,c - travels down polymer to beads 1,2,3.

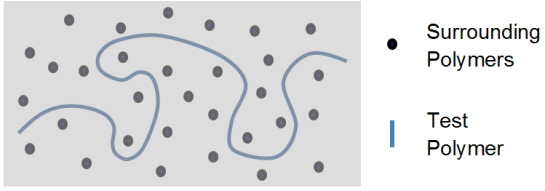


Figure 6: Copied from [3, 14], A 'trace' polymer observes other polymers as entanglement points that constrain it to a tube.

tracer polymers treated as a single backbone chain with no branching; with only two free ends.

In the Rouse model the beads can be connected to their adjacent neighbours either in a straight line, or via a kink in their distribution.[3] For simplicity, extreme kinks are referred to as hairpin bends as shown in figure 5. Internal friction of the polymer acts like tension along the polymer and has a chance to stretch the entropically favourable kinks.[3, 4] The Rouse model assumes that the velocity of the polymer does not cause a reactive backflow from the substance it is diffusing through, and it does not address chain knots. [3]

De Gennes describes the additional effects considered by the reptation model over the Rouse model as accounting for test polymer entanglement with neighbouring polymers, by the addition of a constraining tube around the tracer polymer, demonstrated by figure 6.[3, 14] Once 'felt' by the polymer, the tube reduces the lateral motion of the springs connecting the monomers, as the surrounding polymers would do.[1] The polymer may diffuse away from its original tube or can be drawn through the walls of tube when under collapse, in reality the tube size is not fixed as any other nearby polymers are also diffusing. For polymers in their early stages of diffusion, some are unaware of the tube and therefore do not obey the reptation models constraints within those times.[3, 6]

Take the example of a tracer polymer amongst smaller polymers that are causing entanglement and hence obeying the reptation model. For a tracer polymer of very long length, $N \to \infty$, the polymer melt approaches being comparable to a solvent of small molecules obeying Stokes-Einstein law. This is a result of tube renewable not being observable to a long tracer polymer as all of the smaller polymers are diffusing much faster. [3] The natural conformation of a polymer can be understood in the form of polymer entropy as defined in equation 7 from which free energy and therefore force can be approximated. Any chains stretched have a restoring force to return them to their favourable conformation, and therefore reduce entropy. In an extended reptation model this can be thought as the shape the tracer polymer finds it easiest to exist amongst other polymer and the nanoparticles within a polymer melt. [3, 6]

$$S(\mathbf{R}_e) = -\frac{k_B {\mathbf{R}_e}^2}{2Nb^2} + constant \quad [3, 8] \tag{7}$$

2.2.1 Equilibrium and sub-diffusive behaviour

The Rouse time is the longest time taken for a free chain to relax to its equilibrium conformation. The Rouse time is of the order of time it takes for a hairpin to diffuse down the entire polymer like that shown by procession in figure 5.[3, 1] This time shall be referred to the approximate time

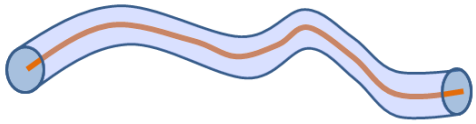


Figure 7: A polymer within its reptation tube.

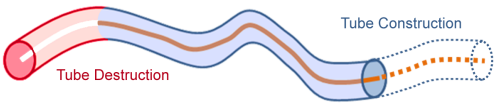


Figure 8: As the polymer diffuses out the initial tube it creates new tube length, the tube at its previous position is destroyed.

to reach sub-diffusive behaviour T_{sub}. This is where the polymer diffuses in a state of equilibrium, but it is not yet free of its initial set of entanglement points. A previous simulation completed by A. Baumgärtner et al. [13] uses this approximate time as a zero point, and starts measuring the diffusion of polymers for $t > T_{sub}$, where the polymer will be collapsed to resemble figure 3.[13]

Conveniently, T_{sub} can be converted to Monte Carlo time $T_{sub}^{(MC)}$ and has been found to obey the following asymptotic relationship for polymer chain lengths $N > 10^3$, where longer polymers tend to;

$$T_{Sub}^{(MC)} = 1.66N^2 \quad [13] \tag{8}$$

2.2.2 Diffusive behaviour

Figures 7 and 8 demonstrate how the polymer within the tube will tend to diffuse out of the tube, destroying the tube at its previous position and constructing a new tube, the time for which a polymer takes to create an entire new tube is known as the polymers reptation time,τ_t. [3, 13] This process is also referred to as tube renewal and is and is quantified by De Gennes as;

$$\tau_t \approx \frac{L^2}{D_{tube}} \approx \frac{NL^2}{D} \quad [3] \tag{9}$$

Where D_{tube} is the diffusivity of the tube, D is the diffusivity of the polymer and L is the length of the tube which scales linearly with N.

This implies the following relationship;

$$\tau_t \propto TN^3 \quad [3] \tag{10}$$

However it is more commonly experimentally observed to follow a power law near $\sim N^{3.4}$.[3, 7] Understanding when the reptation occurs along the timescale T of the polymer is important, as after reptation the polymer has lost memory of its previous conformation, it is an indication that the polymer is starting to reach full centre of mass diffusivity. The entire polymer is free of its initial constraining points. Tube renewal, or full diffusivity, is achieved when the radius of gyration matches that stated by equation 6.

Within previously simulated random walks completed by A. Baumgärtner et al. the length of new reptation tube built and destroyed by the simulated polymers is denoted L. After reaching full diffusivity at time T_{Dif}, L is given by;

$$L(T_{Dif}) = \frac{N}{2} \quad [13] \tag{11}$$

Finally, previous simulations find T_{Dif} also obeys an asymptotic limit for large N polymers in Monte Carlo time steps;

$$T_{Dif}^{(MC)} = 2.62N^3 \quad [13] \tag{12}$$

3 The computational model

The C++ programs constructed for this simulation can be found in GitHub repository;

`https://github.com/Mike-Stringer/Nanoparticles.git`

Appendices A and B contain a flow chart and printed source code of the simulation respectively.

3.1 Previous computational models

The Simulation completed by A. Baumgärtner et al. records the diffusivity for several N using a simulation for a coarse bead reptation model that is described in this section. [13]

3.1.1 Computational diffusion

In this model the diffusivity of a substance is calculated by considering the average centre of mass movement of polymers from initial equilibrium positions. The development of centre of mass movement R_{cm}^2 over time is given by the key equation 13 and is discussed further in section 5.4.

$$< ((R_{cm}(t^{(MC)}) - R_{cm}(T_{Sub}^{(MC)}))^2 > \xrightarrow{T_{MC}\to\infty} DT_{MC} \quad [13, 7] \tag{13}$$

With units in terms of monomer length b. The results shown in figure 9 are all for free polymers experiencing centre of mass diffusion, the diffusion coefficients fit a line as stated in equation 14.[13] They provide a template for which to match results to the free version of the random walk simulation completed for this project, as discussed in section 5.

$$D = 0.04N^{-2}[1 + 50N^{-0.5}], \quad D \sim N^{-2.5} \text{ For t} > T_{Dif} \quad [13] \tag{14}$$

In other experiments D has been commonly observed to follow the a power law of $\sim N^{-2.3}$. [3, 7]

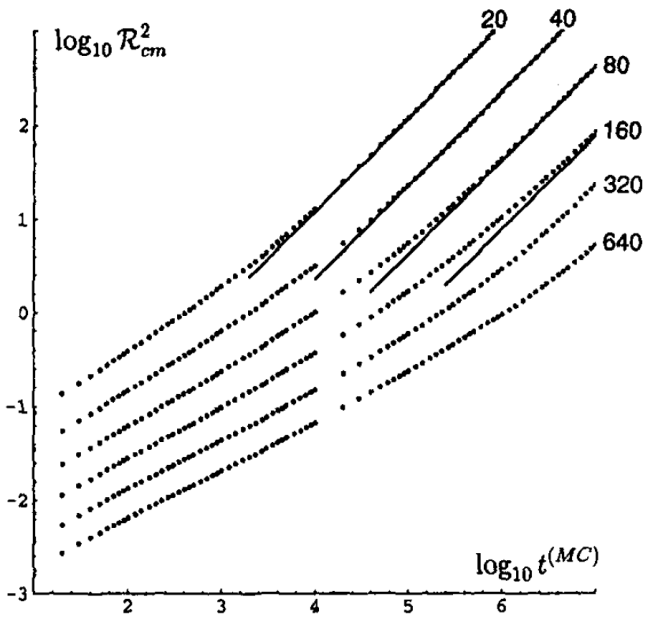


Figure 9: From A. Baumgärtner [13] Diffusive results on simulations recording centre of mass movement R^2_{cm} , diffusion obtained of from straight line gradients using equation 13

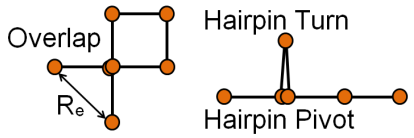


Figure 10: Coarse polymer bead model showing an example of R_e, an allowed overlap and a hairpin bend.

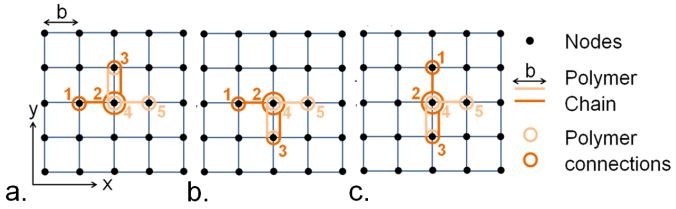


Figure 11: Random walk, The polymer starts at the node labelled 1, finishes at 5 and the polymer overlaps at the node where 2 and 4 are placed, causing a hairpin with the hairpin turn at node labelled 3. Figure part 5a. shows an initial position. 5b. is after polymer hairpin end bead 3 has moved two nodes down from its initial node. 5c. is after polymer start bead 1 has moved one node right and two nodes left.

3.2 A three dimensional random walk

3.2.1 A coarse polymer

As with the simulation completed by A. Baumgärtner et al, the Monte Carlo simulation in this project uses a coarse bead model. A three dimensional grid of nodes spaced by unit cells of size b is constructed, in which the monomers of a polymer, of diameter b, can exists between. The polymers position and movement is tracked by the nodes on which the ends of the monomers are present, known as beads. Therefore, the polymer simply exists as a series of connected beads placed upon nodes, with positions (x,y,z).

The coarseness of the model refers to how the polymer may only move along node points; there is no smaller base unit in which it can change position, for simplicity the base unit is treated as b=1. Pseudo random number generation is used to emulate the stochastic nature of the system and determines the changing of bead positions to other surrounding nodes, and as with the theoretical reptation model, larger values of polymer length N should lead to more statistically significant data. [3, 4]

3.2.2 Pure polymer movement

The thermal stochastic polymer diffusion by Brownian motion is simulated by first randomly picking a polymer bead along its length. That position is allowed to move if it is a free polymer end bead, which as discussed, will only ever be two beads; as no simulated polymers have branching.

As previously stated, the reptation model also allows for the mobility of hairpins. As figure 10 demonstrates hairpins can form and when the bead of the pivot point that is at the hairpin end is selected it may move. These hairpins will allow the polymer to wriggle and diffuse away from their initial position. Once an end or hairpin bead has been picked the direction of motion away from its adjacent node or hairpin base is randomly selected, providing a 1 in 6 chance of being placed in ± x, y or z node. Each bead selection is defined as a random walk and counts as 1/N Monte Carlo time steps; hence N random walks are equivalent to a single Monte Carlo time step.

Finally the reptation model treats polymers overlapping on themselves as allowed phantom entanglement.[3, 8] Any bead along the polymer may occupy the same position as any other bead, aside from the adjacent beads as the polymer monomers are treated as rigid. This emulates the monomer springs cannot collapse to length b = 0. The lack of other form of bead movement, for example corner movement, reflects the constrictions applied by the reptation tube.

An example of allowed random walks within a two dimensional grid is visualised in figure 11.

3.2.3 Nanoparticles

The aim of this project is investigate how the simulated polymers diffusive behaviour changes with the introduction

of nanoparticles. As previously discussed, nanoparticles can be considered of the same order of size and monomer link length, therefore these nanoparticles can be placed individually or as clumps upon grid nodes.

To this end, hard blocking nodes are introduced into the grid, which completely deny any attempt to move a polymer bead upon them. If a bead is selected that can move within a random walk that then tries to move onto a blocked node, that random walk can be treated in two ways. The first method allows random walks to be thrown away and hence 1/N of a time step is lost from that polymers simulation ever time it hits a block. In the other method the target movement node is chosen again until a node that is not blocked is found. The later of these is referred to a forced move simulation and is discounted as a good form of simulation as it does not take into account reduced diffusivity when polymers attempt to move into occupied node.

The introduction of nanoparticles through various simulation algorithms is discussed further in section 6.

3.3 Parallel Computing

Making the simulation as computationally efficient as possible is important for two reasons. The first is that it allows larger polymer lengths N to be run in a reasonable amount of time. The second is too increase the number of polymers that are simulated, as data obtained is averaged over all the polymers.

Both of these reasons contribute to same goal, to increase the accuracy of obtained data and help that data approach statistically significant Gaussian distributions. Running as many polymers as possible reduces the statistical error that is discussed in section 7.1. Longer polymers following the reptation model should tend to reach conformational shapes that themselves fit better to a Gaussian.[8]

It should be noted that to run the program found on GitHub or in appendix B requires Nvidia's Compute unified device architecture (CUDA) and a Nvidia Graphics Card with a minimum of CUDA 2.0 architecture.[15] This program can take up to 6 hours for N = 100 with a Tesla C2075.

3.3.1 CUDA - Compute unified device architecture

Parallel computing is implemented to increase program efficiency and reduce average run times.

The reptation model takes other polymers into account by introducing the constraining tube, computationally this means that no two polymers interact in any way, this is a special computational case known as an embarrassingly parallel problem. Nvidia's CUDA allows each polymer simulation to be calculated by a GPU processing thread on a graphics card. Due to the nature of a how an embarrassingly parallel random walk will require logic statements the advantages of GPU computing do have severe limitations.[15]

GPU threading drastically suffers from divergence, whereby individual threads must compute every possible logic command that any other thread follows within a warp (a set of 32 threads).[15] For this case, the time saved by using GPU computing only comes into fruition when the number of polymers is scaled up. For example, once all 32 threads within any warp are actively all simulating separate polymers the time taken for GPU to complete the simulation will not increase with the addition of more polymers up until the GPUs total capacity. In an experiment to test this, running 1 thread per warp against 32 threads per warp for the same number of threads improved run time by 52.66%.

In order to find a balance between time saved and a large data set, 896 polymers are generated only using 8 active threads per warp. Should any reader want to run the program it is recommended that they possess at least a basic knowledge of CUDA in order to efficiently thread and so as not to overload their GPU beyond its capabilities.

3.3.2 Pseudo-random number generation

The simulation requires a minimum of two random numbers generated per random walk. For full free diffusivity to be observed an upper limit of 1.5×10^7 Monte Carlo time steps is used. Given that the largest value of polymer length simulated in this project is N=100, then the generation of 3×10^9 random numbers is needed for the longest simulations. In an efficiency experiment, the random number generation within the simulation is removed and replaced with a set of quasi-random numbers fed into the simulation, which is only possible for a very short poor accuracy simulation. The program run time is observed to improve by 74.8%.

A large part of improving the simulation comes down to how the program produces random numbers, fortunately CUDA comes with a highly efficient CURAND that can make use of the MRG32K3A, MTGP Mersenne Twister and XORWOW random number generators. These are some of the best available methods of pseudo-random number generation.[16] Since they can be generated on the GPU it allows the simulation to fully run every random walk on the GPU with only one memory copy back to the CPU needed at the end of each simulation. This is advantageous as memory bandwidth between the CPU and GPU can often be the limiting efficiency factor for GPU computing.[15] The distribution of random numbers of this simulation has been confirmed to provide a uniform spread of decisions, in both bead selection and directional random walks.

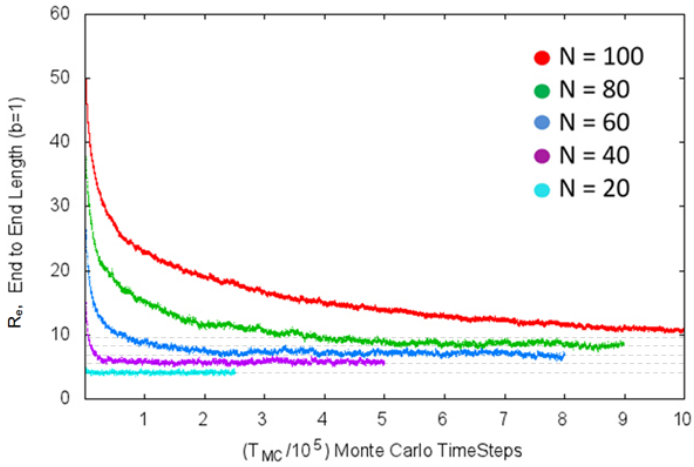


Figure 12: Simulated R_e for polymers of length N=20,40,60,80,100, for 256 polymers in free polymer melt. Dotted lines demonstrate simulated asymptotes.

Polylength, N	End to End Length, R_e		$R_N - R_0$, $\sim N^{0.5}$	Radius of Gyration, R_g		$(Nb^2/6)^{0.5}$	Monte Carlo Time Steps	
	Model		Theoretical	Model		Theoretical	SubDiffuse	Diffuse
100	9.4005±0.0043		10	4.0573±0.0011		4.0825	1600000	2000000
80	8.3603±0.0039		8.9443	3.6209±0.0001		3.6515	750000	1250000
60	7.1415±0.0033		7.746	3.0948±0.0008		3.1623	500000	700000
40	5.8342±0.0028		6.3246	2.5440±0.0007		2.582	200000	250000
20	4.1337±0.0019		4.4721	1.8225±0.0005		1.8257	100000	140000

Figure 13: Tabulated theoretical and simulated values for polymer R_e , R_g . Simulation obtained Monte Carlo time steps $T_{Sub}^{(MC)}$ and $T_{Dif}^{(MC)}$ to be used in all free polymer simulations.

4 Pure polymer diffusion

4.1 Initial polymer placement

The polymers are initialised by one of two methods. In the first method all of the monomers of all of the polymers are all placed on the same free node and then built randomly in the positive direction to stretch out into a three dimensional stepped polymer. In the second method all of the monomers of each polymer are randomly placed on a different free node and then built randomly in the positive direction.

This initialisation involves no hairpin bends and in no way resembles a polymer that is freely diffusing, hence each polymer must be given time to reach equilibrium and begin to diffuse as the reptation model dictates.[3] The variables of conformation that were discussed in section 3 must be used to determine when these polymers reach equilibrium, or as it has also been referred to, sub diffusivity $T_{sub}^{(MC)}$.

4.2 End to end distance

Consider equation 15 that defines the end to end length vector of a polymer, from equation it is known the average value of $\mathbf{R}_e$(x,y,z) from RMS should tend to $N^{0.5}$ for a polymer in equilibrium, therefore the end to end vector can be used to track when a polymer has reached sub diffusivity.

$$\mathbf{R}_e(x,y,z) = \mathbf{r}(x_N,y_N,z_N) - \mathbf{r}(x_0,y_0,z_0) \qquad (15)$$

End to end vector of any polymer can be calculated as so within this three dimensional simulation;

$$\mathbf{R}_e(x,y,z) = \sqrt{(x_N-x_0)^2+(y_N-y_0)^2+(z_N-z_0)^2} \qquad (16)$$

Figure 12 shows the first results of average end to end vector for free polymer diffusion produced by this simulation. For each length the equilibrium asymptotes at an average of around 7% lower than theoretically expected, this could be as a result of the discreteness of the coarse bead model, or it may be to that fact that N is not large enough to produce polymer conformations that approaching realistic systems.

The clarity of what value $T_{sub}^{(MC)}$ takes is reduced with polymer length, but for the most part it can be seen to approach the nature predicted in equation 8. [13]

4.3 Radius of gyration

In addition to the average end to end vector the radius of gyration can be used to confirm subdiffusivity. Recall that radius of gyration is also used in the analysis of experimentally observed polymer diffusion, and is needed to produce the universal curve that relates Φ_{NP}, and D. It is calculated within the simulation using;

$$R_g^2 = \frac{1}{N}\sum_{i=0}^{n}(\mathbf{r}_i - \mathbf{r}_c)^2 \quad [13] \qquad (17)$$

Theoretical and computation values of $\mathbf{R}_e$, R_g, $T_{sub}^{(MC)}$ and $T_{Dif}^{(MC)}$ are listed in figure 13.

4.4 Center of mass movement

After sub diffusion has been reached within the simulation, the position and conformation of the polymer are recorded and stored as a zero point from which to start calculating the centre of mass diffusion. The polymers should undergo some observable change in which their diffusivity begins to scale with that predicted by the reptation model in equation 10.[3] In addition this should occur around the time of full tube renewal for which a theoretical value is predicted by equation 11 and previous computational scaling laws stated in equation 12.

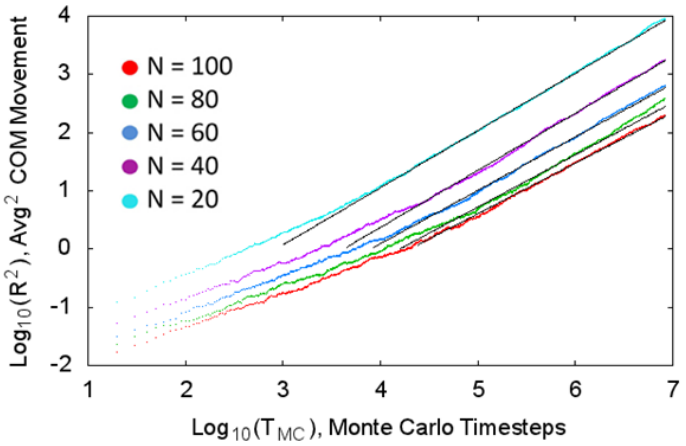


Figure 14: First simulated R^2_{cm} data for free polymer melt of polymers of length N = 20, 40, 60, 80, 100. Diffusion obtained of large $t^{(MC)}$ straight line gradients using equation 13

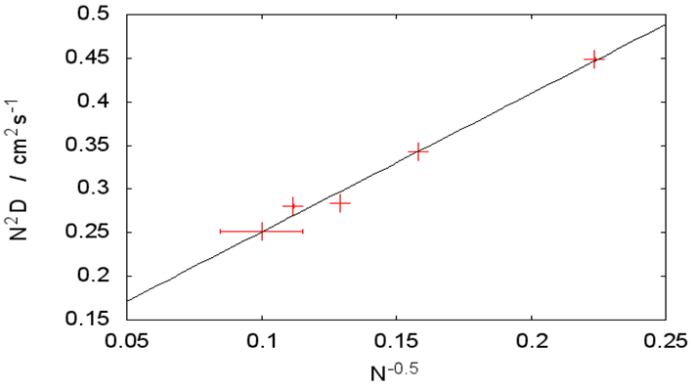


Figure 15: Gradient obtained diffusivity's for all N, after kink in figure 14, follows fit defined by equation 14.

The observable change is referred to as full centre of mass diffusivity, and occurs for all Monte Carlo time steps beyond $T^{(MC)}_{sub}$. It can be used, given that the reptation model holds, to calculate the diffusion coefficient of the polymers by solving equation 13. R^2_{cm} movement is given by;

$$R_{cm} = \frac{1}{N}\sum_{i=0}^{n} \mathbf{r}_i \quad [13] \tag{18}$$

The following preliminary results are from autumn 2014 and show that the simulations calculated diffusive behaviour matches that produced by A. Baumgärtner et al. in their previous computational model for a pure polymer melt. Figure 14 matches figure 9 and figure 15 matches the diffusive fit dictated in equation 14. These diffusive constants are referred to as the pure polymer diffusion coefficients D_0. For N=100, D_0 has large error bars; as these preliminary simulations did not perform enough Monte Carlo time steps to track full diffusivity for a polymer of such long length.

5 Producing Nanoparticles

5.1 Occupation density, block chance and block count

The variable Φ_{NP} is used in this simulation to quantify the volume fraction of nanoparticles that are produced in the algorithms described in this section. Two other variables are also considered to describe the amount of nanoparticles within each simulation.

The first, block chance, takes into account the placement of the polymer amongst the nanoparticles and calculates the average chance that a movement in any direction would result in a block. A density number describing the ratio of block moves to free moves could then be obtained. Block chance therefore can distinguish between volume fractions that are uniform and those that are not, for example a clump of nanoparticles together have only external faces reducing polymer diffusivity whilst the same number of nanoparticles spread out evenly would have all six adjacent nodes causing local reduced diffusivity.

The second, block count is an internal number within the simulation which physically obtains the ratio of free moves to blocked moves. Preliminary results showed, surprisingly, that block count did not match block chance. A possible explanation for this is the polymer has a 'memory', more explicitly the polymer knows that precisely every point along its length is not a block. Since many random walks are actually collapsing breathing modes where the polymer wiggles amongst its own conformation then many random walks actually have no chance or a reduced chance of hitting a blocked node. As is observed, block count should therefore yield a lower ratio of block moves to free moves than block chance. It is decided that the best way to quantify nanoparticles is still volume fraction however block count can be of use for analysis to help produce block algorithms.

5.2 'Real' Nanoparticles or repetitive 'virtual' Nanoparticles

The challenge of adding nanoparticles into the simulation came down to a choice between what shall be referred to 'real' nanoparticles or repetitive 'virtual' nanoparticles. As mentioned previously nanoparticles diffuse extremely little along the timescales of a typical tracer polymer.[2] This makes the assumption that once placed upon nodes, nanoparticles should remain on the same nodes for the entire simulation is valid, and makes preparing the simulation simpler.

Having real nanoparticles would mean that every explicit node in an entire massive grid of nodes could be, prior to the random walk, cast as either a free node or a hard blocking node with a nanoparticle upon it.

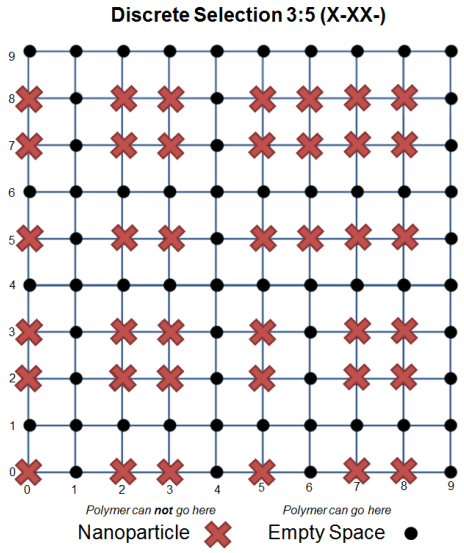


Figure 16: Discretely placed nanoparticles on a repetitive 2D grid of size 5. (X - X X -)

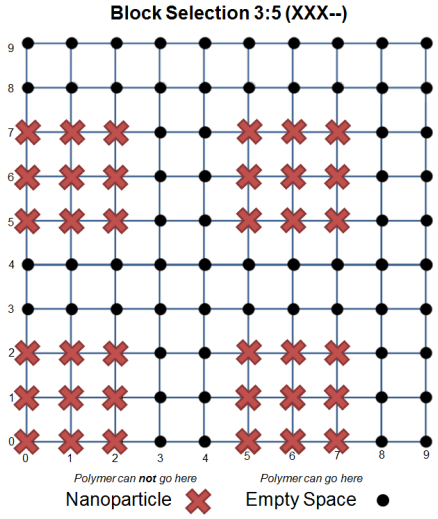


Figure 17: Clumps of nanoparticles (or large nanoparticles) placed on a repetitive 2D grid of size 5. (X X X- -)

The 'Real' model would allow the most flexibility in placement of nanoparticles but would do so at the cost of computational efficiency.

Because the volume fraction of nanoparticles in polymer melts is typically evenly distributed, as proved with cross-sectional images from TEM;[2] it is an acceptable assumption that the nanoparticles can be thrown into the simulation as grid of repetitive virtual particles.

Virtual particles are computationally favourable as they only exist as a particular remainder of 'density grids'. Density size in the distance across in the x, y and z directions which is repeated over, take any potential node coordinate and when divided by the 'density grid' size a particular set of the potential remainders can be detonated as blocked notes, whilst the rest remain as free. Figures 16, 17 and 18 demonstrate several potential two dimensional 'density grids'. Recall that the simulation is three dimensional and has a 'density grid' (volume) for (x,y,z) coordinate spaces.

5.3 Nanoparticle algorithms

Three key algorithms are developed to simulate repetitive 'virtual' nanoparticles.

5.3.1 Discrete placement

Figure 16 demonstrates discrete placement. With discrete placement the nanoparticle algorithm is left un-general and a set of specific logic laws for a chosen repetitive 'density grid' are enforced for every random walk.

5.3.2 Clump placement

Figure 17 demonstrates clump placement. With clump placement the code is generalised so that the 'density grid' can be any size and so that the clumps of nanoparticles can be any size up to ('density grid' size - 1). In addition the coarseness of the polymer can be altered to other base units (e.g. b = 2), allowing simulations where the ratio of nanoparticle size to monomer size is altered. The latter capability of clump placement is extremely computationally expensive as it requires many more nodes to be scanned over for each potential random walk. It is of course possible to create a discrete placement algorithm for any potential clump placement; however the discrete placement algorithm has greater CUDA thread divergence than the clump algorithm.

5.3.3 Pseudo-random placement

Figure 18 demonstrates a potential Pseudo-random placement. This algorithm is essentially an expansion upon the clump placement algorithm that directly replaces clumps of nanoparticles with three arrays of random numbers from 0 to 'density grid' size for the x, y and z directions. This array is varied in size as a measure of volume fraction. It is for this algorithm that all polymers are completely randomly initialised, all three coordinate systems have a different configuration of random blocks, and it is also the only algorithm that averages over multiple polymers melts with the same volume fractions but different configurations. Finally, it is still relatively similar to the computationally efficiency of clump placement.

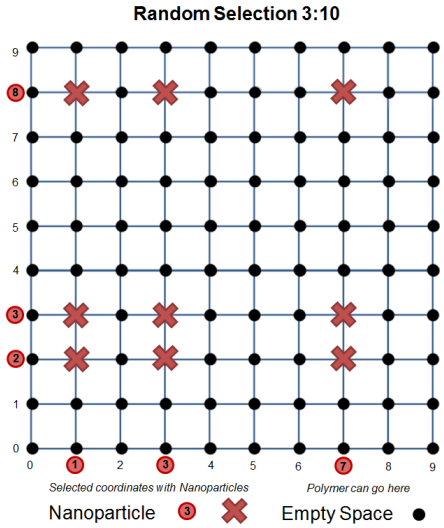


Figure 18: Randomly placed nanoparticles on a repetitive 2D grid of size 10. (3 in 10)

6 Model Analysis

6.1 Data distribution and errors

The reptation model only produces truly Gaussian statistics for very large N.[3, 4, 1] The distribution of all obtained data will follow the same spread as the example for R_e probability in equation 19;

$$P(\mathbf{R}_e, N) = \left(\frac{2\pi N b^2}{3}\right)^{-\frac{3}{2}} e\left(-\frac{3\mathbf{R}_e^2}{2Nb^2}\right) \quad [3,\ 4,\ 1] \qquad (19)$$

Therefore, Gaussian statistics are not achievable within these coarse polymer simulations. In fact, due to the discreteness of data and the fact that all recorded variables are average magnitudes and must be positive in value; the spread of data across many polymers resembles that of a Poisson distribution, particularly for smaller polymers.

The distributions of snapshots of obtained centre of mass R_{cm}^2 for one time step across 896 polymers are presented in figures 19, 20 and 21, where the snapshots are taken just after sub diffusion, after full centre of mass diffusion, and after 1.5×10^7 Monte Carlo time steps respectively. All errors for the centre of mass movement that are calculated in this section are statistical errors obtained from the skewed variance of the large number of polymers simulated, statistical errors are of the order $\sim \pm 3\%$ for R_{cm}^2. The distribution in figure 19 has mean $R_{cm}^2 = 3.48$ (base unit squared), standard deviaton $\sigma = 3.60$, $\Delta\ R_{cm}^2 \sim 3.45\%$. Figure 20 has mean $R_{cm}^2 = 349.78$, $\sigma = 280.29$, $\Delta\ R_{cm}^2 \sim 2.68\%$.

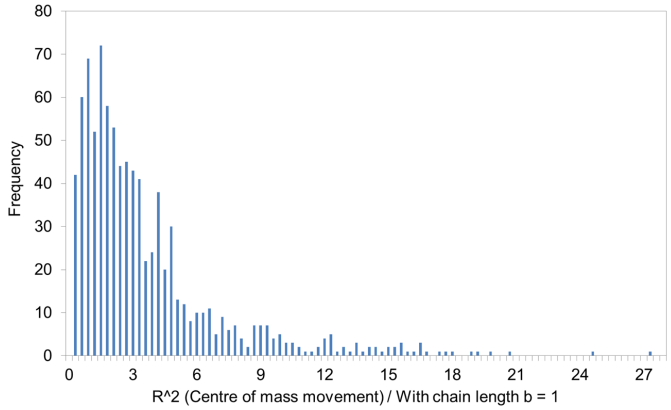


Figure 19: A frequency distribution of R_{cm}^2 data for a short polymer N=20 and just after R_{cm}^2 has been set to zero at $T_{Sub}^{(MC)}$.

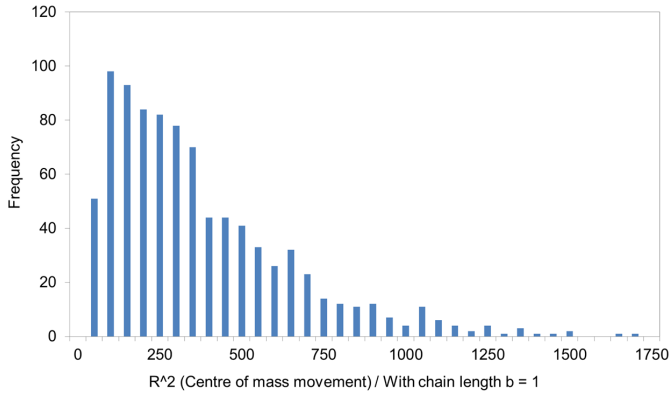


Figure 20: A frequency distribution of R_{cm}^2 data for a short polymer N=20 and after polymer simulation is beyond $T_{Dif}^{(MC)}$.

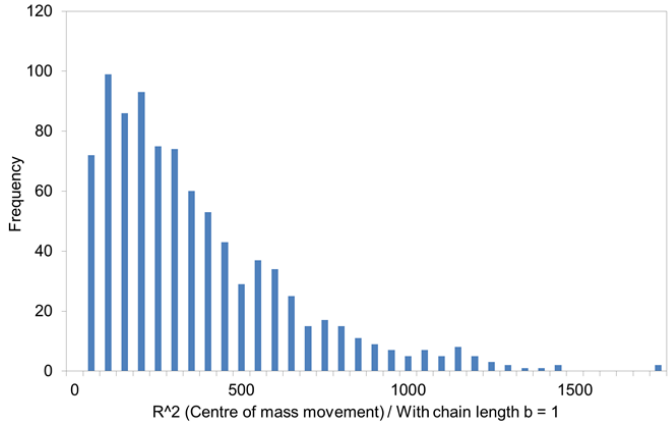


Figure 21: A frequency distribution of R_{cm}^2 data for a long polymer N=100 and after 1.5×10^7 Monte Carlo time steps.

Placement Config.	NP Density	Volume fraction, Φ_{NP}	N=100 Diffusivity	Approx. ID/2R_g
RAND	1 in 10	0.001	(2.664±0.003)x10^{-5}	1.130
RAND	1 in 7	0.003	(2.622±0.002)x10^{-5}	0.751
RAND	1 in 5	0.008	(2.616±0.003)x10^{-5}	0.505
RAND	2 in 9	0.011	(2.708±0.001)x10^{-5}	0.444
RAND	1 in 4	0.016	(2.769±0.003)x10^{-5}	0.379
RAND	2 in 7	0.023	(2.655±0.002)x10^{-5}	0.315
X--X--X-	3 in 8	0.053	(2.735±0.003)x10^{-5}	0.213
X-X--	2 in 5	0.064	(2.686±0.002)x10^{-5}	0.192
X-	1 in 2	0.125	(3.492±0.002)x10^{-5}	0.125
X-XX-	3 in 5	0.216	(3.606±0.002)x10^{-5}	0.086
FREE	--------	--------	(2.662±0.002)x10^{-5}	----------

Figure 22: For N=100. Table of complete simulated nanoparticle production methods with respective volume fraction Φ_{NP}. Values for diffusivity are obtained from gradients taken from $10^{6.1} \rightarrow 10^{7.1}$, and radius of gyrations are averaged over the same period. ID is approximated as the average distance to the nearest neighbour. Units of diffusivity are given by $m^2 s^{-1}$ with unit polymer length b=1. Values for D/D_0 that are discussed and presented in figure 36 can be obtained from these results.

The deviation of data tends to decreases after more Monte Carlo time steps have passed. The best statistical data is therefore expected for the largest N simulations ran for the longest time, figure 21, a snapshot distribution of a N=100 simulation ran for the upper limit of 1.5×10^7 Monte Carlo time steps has a mean $R^2_{cm} = 340.14$, $\sigma = 282.37$, $\Delta R^2_{cm} \sim 2.77\%$, the tightest distribution observed for any full N = 100 simulations, but less so than N=20 after $T^{(MC)}_{Dif}$.

6.2 Results

The following volume fraction simulations as presented in figure 22 are run for 1.5×10^7 Monte Carlo time steps for polymer lengths N = 20, 40, 60, 80, 100. All of the explicit distributions of nanoparticles are completed using the discrete placement algorithm, whilst several 'RAND' simulations are completed using the Pseudo-random placement algorithm.

To reiterate, the diffusivity of the final results is determined from the gradient after a turning point is observed in the R^2_{cm} data, confirming tube renewal. This turning point, $T^{(MC)}_{Dif}$, is observed at $\sim 10^{6.1}$ for N=100. R_g and R^2_{cm} are recorded from $T^{(MC)}_{Dif}$ onwards. The results which are stored in figure 22 are discussed in the following sections.

ID, inter-particle distance can be calculated by two methods; in the first method the approximated average distance from each nanoparticle to it's nearest neighbour is taken as ID(again, in terms of base unit b=1). This value is not averaged over next nearest neighbours or beyond since the primary nanoparticle confining the conformation of the polymer is assumed to be the closet blocking node. In the second method equation 4 is used, where each nanoparticle is assumed to occupy its own distinct volume. There is little scaling difference between these two methods. The first method is used calculate values displayed in figure 22.

Polylength, N	End to End Length, R_e		$R_N - R_0$, ~ $N^{0.5}$	Radius of Gyration, R_g		$(Nb^2/6)^{0.5}$	Monte Carlo Time Steps	
	Dense		Free	Dense		Free	SubDiffuse	Diffuse
100	9.1381±0.0042		9.4005±0.0043	3.9649±0.0011		4.0573±0.0011	2000000	2500000
80	8.3402±0.0040		8.3603±0.0039	3.6101±0.0001		3.6209±0.0001	1000000	1500000
60	7.1728±0.0032		7.1415±0.0033	3.1085±0.0008		3.0948±0.0008	750000	1000000
40	5.8642±0.0027		5.8342±0.0028	2.5494±0.0007		2.5440±0.0007	250000	300000
20	4.1166±0.0020		4.1337±0.0019	1.8159±0.0005		1.8225±0.0005	125000	150000

Figure 23: Tabulated simulated values for polymer R_e , R_g comparing $\Phi_{NP} = 0.5$ to free polymer values. Also new diffuse and sub diffuse time used in simulations to ensure equilibrium and tube renewal.

For all of the following results (aside from 2D compare - figure 25) it is convenient to graph the statistical errors as the vertical spread of results.

6.2.1 Confirming sub-diffusivity

In the case of high nanoparticle volume fractions it is expected the polymers will reach sub diffuse behaviour after a different number of Monte Carlo time steps compared to that of a free polymer. Preliminary tests that collect both the end to end vector and radius of gyration data ensure that asymptotic values have been reached and sub diffusivity is occurring. In addition values for which the radius of gyration and end to end vectors settle at have been observed to change. Observed values of the most extreme case of simulated $\Phi_{NP} = 0.5$ are listen in figure 23.

It is observed that except for the longest polymer lengths there is no significant change in either of the conformation values R_e or R_g. For the case example of N=100 R_e and R_g are 2.79% and 2.28% lower than a fully free diffusing polymer respectively. This constraining effect may only be observable for high N in the coarse polymer reptation model but there is not enough data to confirm this further.

6.2.2 Initial reactions and two dimensional comparison

Figure 24 presents a set of results for high volume fraction $\Phi_{NP} = 0.125$ (X-) configuration with 1 nanoparticle node and 7 free nodes in every repeated volume of 8 nodes. For every polymer length N the diffusivity is clearly increased from free polymer diffusion of the same length polymer, this does not match what is expected from experimental data [2, 6]. Just from these of results alone it is clear that the coarse polymer model has some significant factor affecting diffusivity that is not present in experiment data, or that the coarse polymer model approximates polymer movement too discreetly, particularly for small N. There is however, no

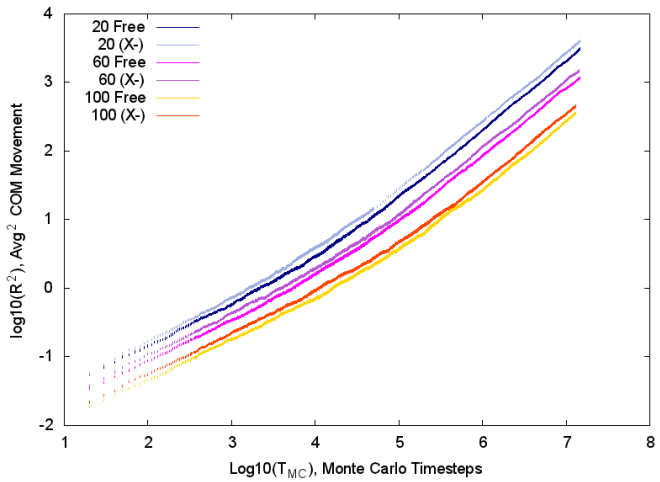


Figure 24: A R^2_{cm} comparison for N = 20, 60, 80 for free and (X-) $\Phi_{NP} = 0.125$ diffusion, showing large increase in gradient ((X-) higher up) and therefore higher diffusivity for more dense systems, the opposite of what is experimentally observed.

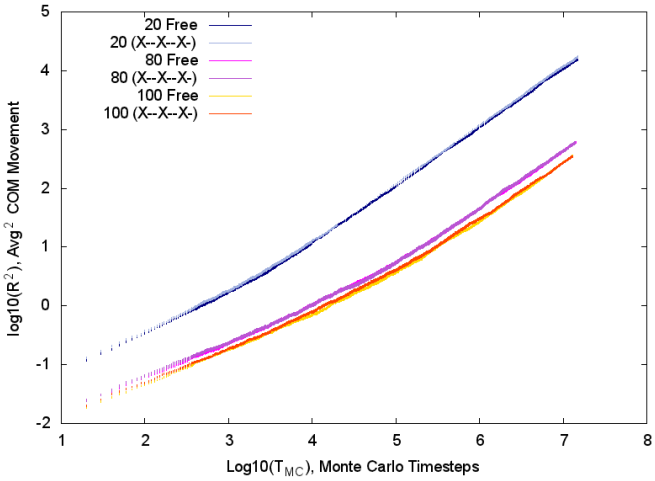


Figure 26: A R^2_{cm} comparison for N = 20, 80, 100 for free and (X–X–X-) $\Phi_{NP} = 0.053$ diffusion. First indication that the effect of low Φ_{NP} simulated nanoparticles is unclear and might decrease diffusivity below free polymers.

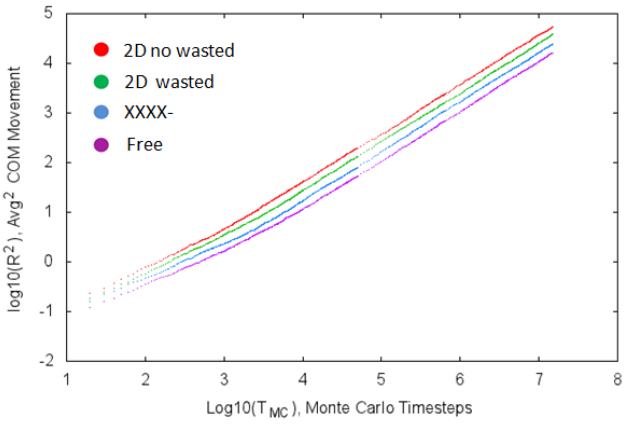


Figure 25: A R^2_{cm} comparison for the following N = 20 simulations; A two dimensional free random walk, a two dimensional free random walk with wasted time steps attempting to move in the z direction, a (XXXX-) configuration random walk and a standard free polymer melt random walk. Extreme Φ_{NP} simulations approach the diffusion coefficients of a 2D system.

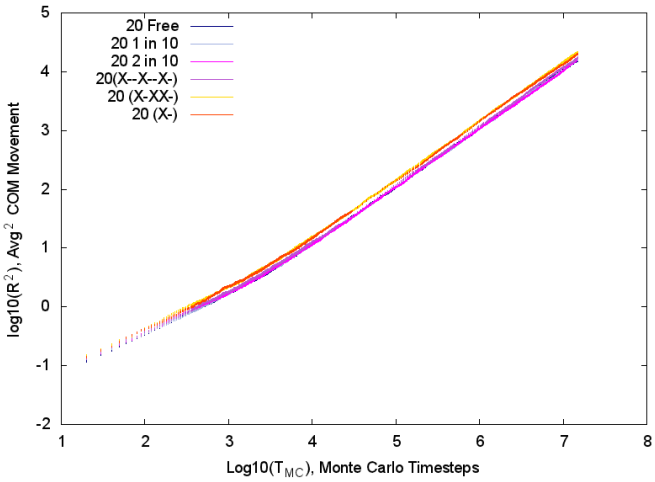


Figure 27: A R^2_{cm} comparison for N = 20, for free and various nanoparticle volume fractions, showing spread of diffusivities with clear outliers of (x-) and (X-XX-) whilst less dense (1 in 10) and (2 in 10) lie close or even below freely diffusing.

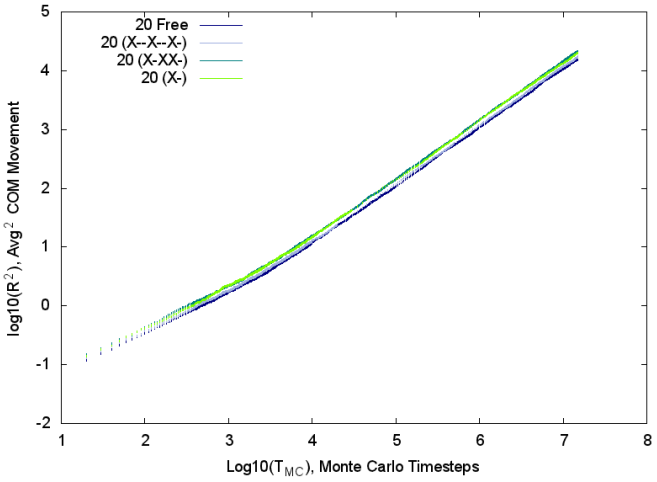


Figure 28: A R^2_{cm} comparison for N = 20, clarifying the discrete algorithms with volume fractions $> \Phi_{NP} = 0.053$ all appear to have diffusivity higher than the free simulations.

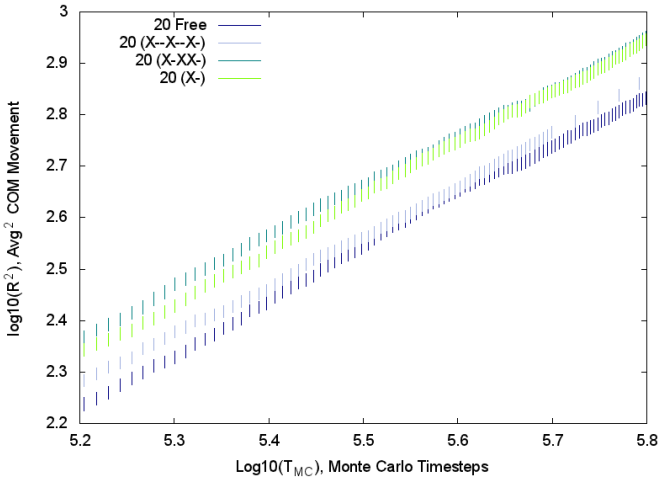


Figure 29: A R^2_{cm} comparison for N = 20, In conjuncture with diffusive values printed in figure 22 this close up with a tight Monte Carlo time step time range clearly shows that all of the discrete algorithms considered lie above the free polymer.

observable change over the multiple polymer lengths N. All $\Phi_{NP} = 0.125$ simulations have increased diffusivity beyond free polymer movement. Even for the maximum polymer lengths of this simulation N=100, the increase in diffusivity remains present and indistinguishable from the increased observed in other polymer lengths.

A possible explanation for higher mobility at higher nanoparticle densities might be the increased frequency and rapidity of hairpin formation in a constrained system. Consider a two dimensional random walk, it reaches equilibrium in less Monte Carlo time steps and has greater diffusivity than a 3D random walk due to an increased number of hairpin bends, such a system might be comparable to a high density occupation. Figure 25 contains a set of N=20 simulated tests to compare the high density (XXXX-), $\Phi_{NP} = 0.512$ 3D random walk against two 2D simulations; one where the random walks in the three dimension have a probability of being produced but are thrown away (2D wasted) and one where a pure 2D random walk for a polymer is simulated. In the latter case the chance for polymer to fold in on itself and produce a hairpin is greatly enhanced as there are less free spaces with no polymer monomers present. The results demonstrate that within the simulations, more constrained systems approach the diffusivity of a 2D system.

In order to prove if the discrete algorithm for nanoparticle production is working, a greater range of discrete volume fractions is tested, the minimum of which is presented in figure 26, with $\Phi_{NP} = 0.053$, at this low density the diffusivity of the polymers are close to that of a free polymer of the same length. In order to investigate if any configurations of nanoparticles actually cause a decrease in diffusivity as is predicted by experimental data, a closer examination of N=20 and N=100 and simulations is needed, including calculated diffusion coefficients.

6.2.3 Final diffusion variation observed

For the following sections fraction densities (a in b) will be quoted, where a is the number of random numbers in each axis (of x, y and z) out of a repeated volume of b^3.

A spread of volume fraction data is collected for N=20, introducing the aforementioned pseudo random algorithm to produce nanoparticles. It is important to mention that tests completed after the full length simulations confirmed that the discrete algorithm of nanoparticles produced the same observed increase in diffusivity for high volume fractions as the pseudo random algorithm. It just developed that the simulations completed later, of smaller densities, were completed using the newer pseudo random algorithms.

Figure 27 contains all of the N=20 simulations, and shows the first indication of some low (1 in 10), $\Phi_{NP} = 0.001$ and (2 in 10), $\Phi_{NP} = 0.008$ algorithms near or below the free polymer diffusion. Longer polymers of length of N=100, with more statistically significant sets of data; are used to analyse the effect of lower volume fractions, with explicit calculation of diffusivities.

Figure 28 and 29 once again demonstrate that the gradient determined diffusion coefficients are definitely greater than the free polymer coefficients of the same length for all $\Phi_{NP} > 0.053$.

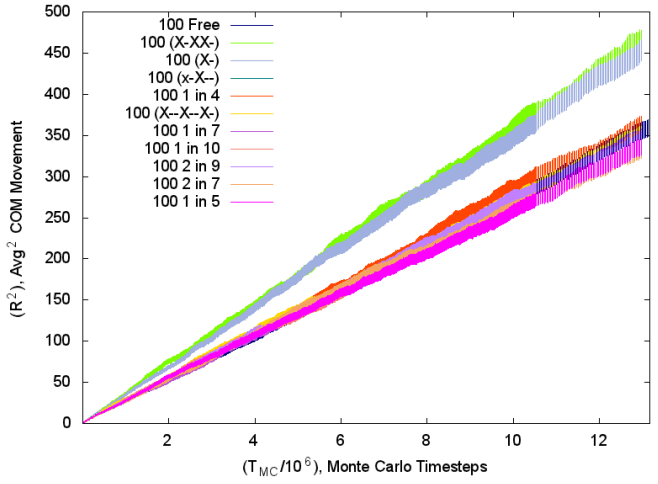


Figure 30: A none log R^2_{cm} comparison for all N = 100, for free and various nanoparticle volume fractions to show distribution of entire set of full length results and begin to show the lower diffusivity observed in (1 in 5), $\Phi_{NP} = 0.008$, (1 in 7), $\Phi_{NP} = 0.003$, (2 in 7), $\Phi_{NP} = 0.023$ (and ambiguously (1 in10), $\Phi_{NP} = 0.001$).

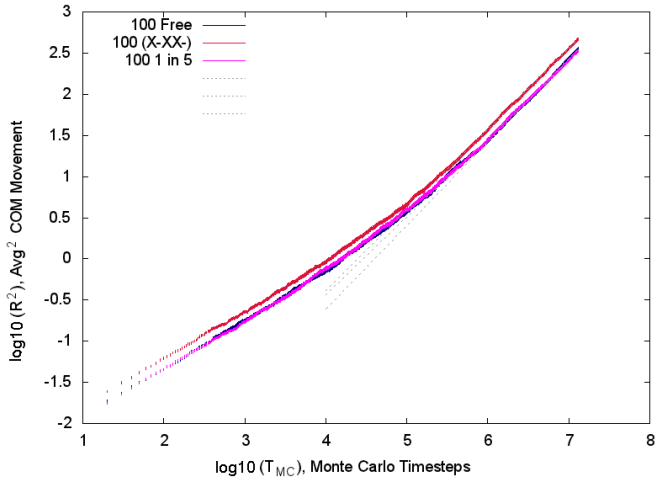


Figure 31: A R^2_{cm} comparison for N = 100, with (1 in 5), $\Phi_{NP} = 0.008$, (X-XX-) $\Phi_{NP} = 0.216$ and free with straight line gradients to demonstrate obtained diffusivities.

Figure 30 contains a full set of none logarithmic data showing the clear divergence in gradients between volume fractions .All data is used to produce diffusivities, which shall be discussed in the final section, however an example of some fitted diffusivity data is displayed in figure 31.

The data in figures 32, 33, 34 and 35 is presented over a much tighter range of $10^{6.9} \rightarrow 10^{7.12}$ Monte Carlo time steps, this used to closely examine the comparable diffusion of the nanoparticle volume fractions that are calculated to lie close to the free polymer diffusion for N=100, $D_0 = (2.662 \pm 0.002) \times 10^{-5}$.

Out of configurations (1 in 4), (1 in 5), (1 in 7), (1 in 10), (2 in 7) and (2 in 9), the lower diffusion coefficients are calculated to be; [1 in 5, $\Phi_{NP} = 0.008$, D = $(2.662 \pm 0.002) \times 10^{-5}$], [1 in 7, $\Phi_{NP} = 0.003$, D = $(2.662 \pm 0.002) \times 10^{-5}$], and [2 in 7, $\Phi_{NP} = 0.023$, D = $(2.662 \pm 0.002) \times 10^{-5}$]. These lower diffusivities can be seen, whereas the other volume fraction are not as clear aside from (1 in 4), where the polymers clearly have higher diffusion than the free polymers.

If the observed decrease in diffusivity is correct, and if the assumption that higher volume fractions eventually cause higher diffusivity because of the increased frequency of hairpins, then we expect to see a turning point. This turning point is where the decrease in diffusivity due to the loss of Monte Carlo timesteps wasted hitting blocked nodes is overcome by the increase in diffusivity due to the boosted mobility of a polymer with more hairpins to wriggle upon.

Such a turning point is not observed. Whilst there is a minimum at (1 in 5), $\Phi_{NP} = 0.008$ with diffusive constant D = $(2.662 \pm 0.002) \times 10^{-5}$, we expect lower volume fractions to return back to the diffision of a fully free polymer as their are not enough blocks to produce an observable effect and we expect higher volume fractions to start increasing in diffusivity due to hairpin induced mobility.

Figure 33 breaks this rule as (1 in 7) with $\Phi_{NP} = 0.008$ diffusion coefficient lies below (2 in 9) with $\Phi_{NP} = 0.011$ despite being both after the observed minimum. Figure 35 again has two configurations the wrong way round as 2 in 7 with $\Phi_{NP} = 0.023$ diffusion coefficient lies below (1 in 4) with $\Phi_{NP} = 0.016$.

There is an argument to be made for the data being too sporadic as there are observable fluctuations in all of these data sets, and yet the calculated diffusion coefficients reinforce that the order of the diffusion coefficients beyond the observed (1 in 5) diffusion minima do not ascend in the order of Volume fraction well within statistical error.

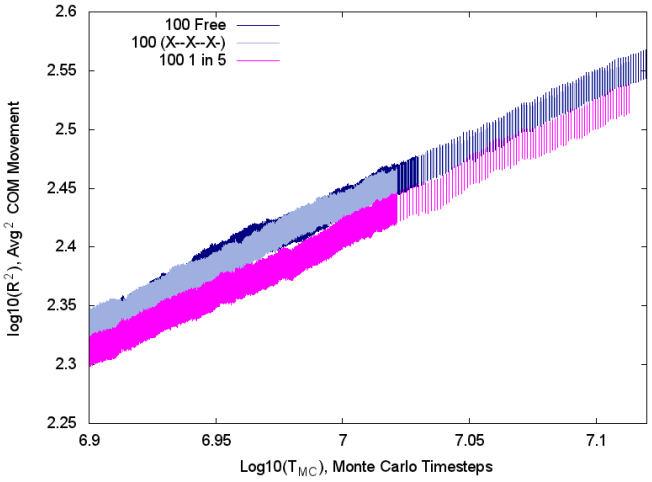


Figure 32: A R^2_{cm} comparison for N = 100, In conjuncture with diffusive values printed in figure 22 this close up with a tight Monte Carlo time step time range clearly shows that (1 in 5) $\Phi_{NP} = 0.008$ has slower diffusion than a freely diffusing polymer as is experimentally observed.

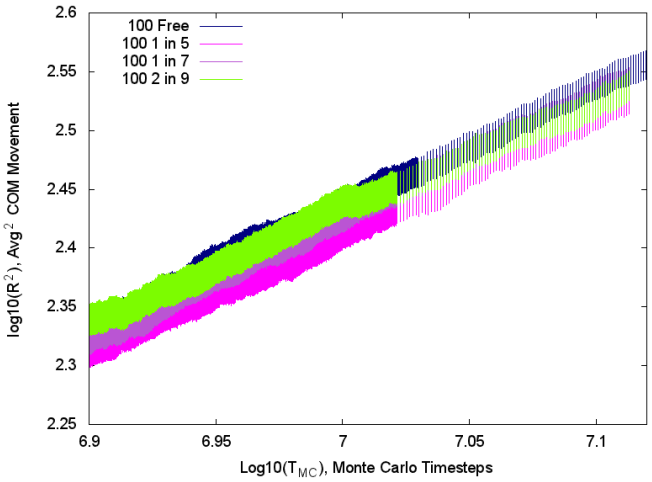


Figure 34: A short range of Monte Carlo time steps R^2_{cm} comparison for N = 100, demonstrating (1 in 7), $\Phi_{NP} = 0.003$ also appears ambiguous compared to those previously studied. Both (1 in 7) and (2 in 9) do appear to lie below free diffusion.

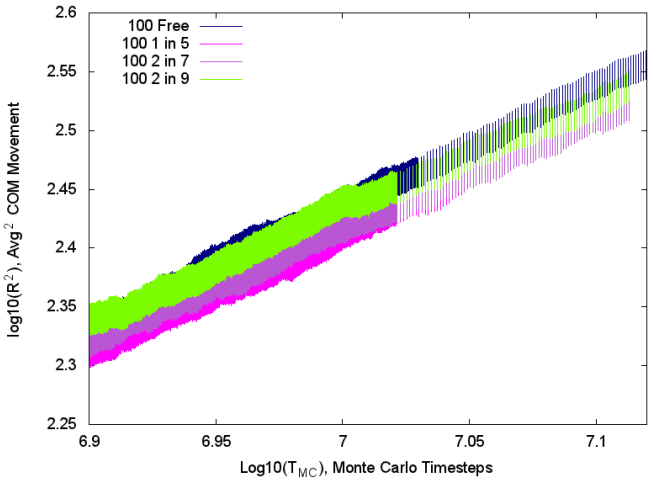


Figure 33: A short range of Monte Carlo time steps R^2_{cm} comparison for N = 100, (1 in 5), $\Phi_{NP} = 0.008$ and (2 in 7), $\Phi_{NP} = 0.023$ have lower diffusivity than a free polymer whilst (2 in 9), $\Phi_{NP} = 0.011$ is ambiguous.

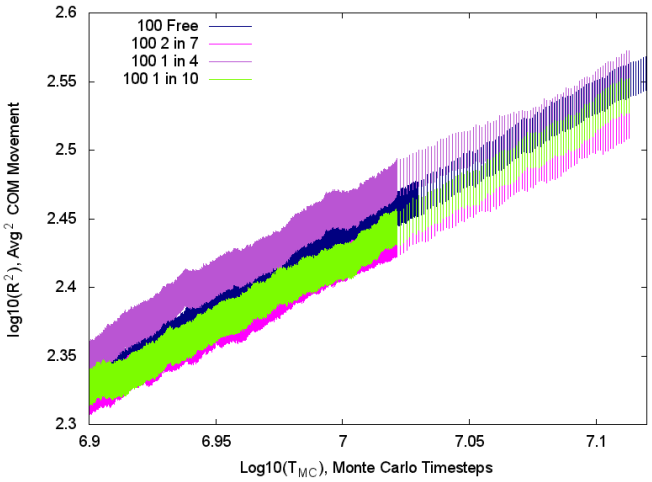


Figure 35: A short range of Monte Carlo time steps R^2_{cm} comparison for N = 100, demonstrating (1 in 4), $\Phi_{NP} = 0.016$ and (1 in 10), $\Phi_{NP} = 0.001$. (1 in 4) lies above free polymer diffusion whilst (1 in 10) lies below.

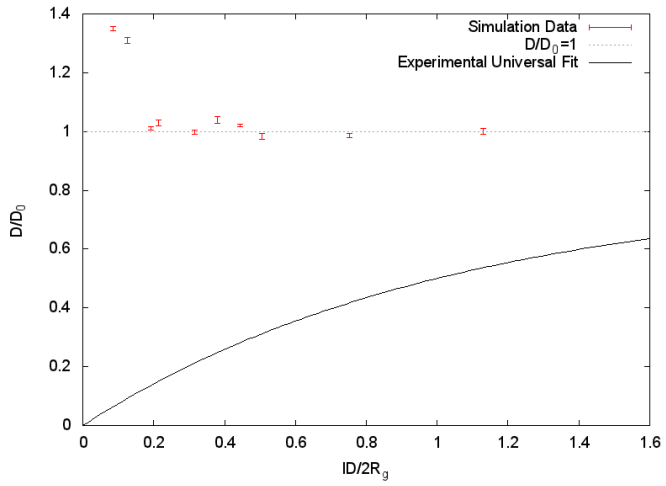


Figure 36: Simulated D/D_0 against $ID/2R_g$ (values printed in table 22 for N=100 polymers with densities previously discussed. Only (1 in 5) $\Phi_{NP} = 0.008$, (1 in 7) $\Phi_{NP} = 0.003$, (2 in 7) $\Phi_{NP} = 0.023$ lie below D/D_0 as their diffusivity is observed to fall with the addition of nanoparticles. For all other densities diffusivity actually increases, with an exponential tendency towards higher diffusivity for higher nanoparticle volume fractions. Experimental universal curve is provided for comparison.

6.2.4 Experimental against simulation

Figure 36 brings together all of the diffusion coefficients for N=100 simulations across all of the volume fractions. D/D_0 is plotted against $ID/2R_g$ so that only the three nanoparticle configurations discussed with decreased diffusivity lie below the $D/D_0 = 1$.

For the rest of the data there is some ambiguity when near volume fraction closest to the observed minimum diffusivity for (1 in 5), $\Phi_{NP} = 0.008$, $D = (2.616 \pm 0.003) \times 10 - 5$ or $D/D_0 = 0.983 \pm 0.001$; but for $\Phi_{NP} > 0.064$ diffusivity is observed to always increase with volume fraction.

None of the results match the experimental universal curve. [2, 6]

7 Final Remarks

The inability for the coarse polymer reptation simulation to match previous experimental results could be due to several reasons. Firstly the increased mobility of a constrained polymer due to a greater frequency of hairpin turns might be causing the simulation to rapidly increase diffusivity away from the universal curve. Instead of nanoparticles decreasing diffusivity, hairpins cause an increase in diffusivities that dominates for $\Phi_{NP} > 0.008$. Evidence that supports this explanation comes from increasingly constrained simulated polymers tending towards the diffusivity of a two dimensional random walk, where the diffusivity decrease due to time wasted hitting blocked nodes is less than the increase in diffusivity due to the extra motion in two directions.

Secondly the length of the polymer never exceeded N=100 in these simulations, whilst this reduced statistical errors down to a minimum of $\Delta\ R^2_{cm} \sim 2.77\%$, it is possible this length is not long enough to produce statistically meaningful diffusion coefficients. More likely however is that the coarse discrete nature of the polymer in conjunction with a maximum of N=100 produce a system that is simply unrealistic. A computationally expensive version of this simulation is designed to scale the coarseness of the polymers in ratio with the placement of nanoparticles, and could prove to be a interesting direction for further research.

Finally, the placement of the nanoparticles by repetitive 'density grids' may still be a successful and efficient method of placing nanoparticles, the psuedo-random algroithm is efficient and the CUDA implementation does have a potential to have the number of simulated polymers averaged over dramatically increased. The distribution of experimentally observed nanoparticles is typically very even so the current implementation of nanoparticles should be valid, however, the alternate implementation of explicit 'real' nanoparticles described in section 5.2 could make the simulation more realistic.

No conclusion could be reached on if this kind of coarse reptation simulation will always tend to reach an exact minimum value for diffusivity in low volume fractions before increasing with greater volume fractions, always observed here as for $\Phi_{NP} > 0.064$. The low volume fraction configurations should be studied with greater statistical significance to determine the true origin of the observed diffusivity minima and scaling beyond it. This may provide a method for which to simulate polymers that are more similar to those observed experimentally, and that more closely follow the experimentally universal curve of D/D_0 against $ID/2R_g$.

References

[1] M. Rubinstein and R. Colby, "Polymer physics, oxford university press," *New York*, 2003.

[2] S. Gam *et al.*, "Macromolecular diffusion in a crowded polymer nanocomposite," *Macromolecules*, vol. 44, no. 9, pp. 3494–3501, 2011.

[3] P.-G. de Gennes, "Reptation of a polymer chain in the presence of fixed obstacles," *The Journal of Chemical Physics*, vol. 55, no. 2, pp. 572–579, 1971.

[4] M. Doi, *The theory of polymer dynamics*. oxford university press, 1988, no. 73.

[5] J. Crank, *The mathematics of diffusion*. Oxford university press, 1979.

[6] J. Choi *et al.*, "Universal scaling of polymer diffusion in nanocomposites," *ACS Macro Letters*, vol. 2, no. 6, pp. 485–490, 2013.

[7] J. Deutsch and T. Madden, "The diffusion coefficient of a reptating polymer," *The Journal of chemical physics*, vol. 91, no. 5, pp. 3252–3257, 1989.

[8] R. A. Jones, *Soft condensed matter*. Oxford University Press, 2002, vol. 6.

[9] M. Mu *et al.*, "Polymer diffusion exhibits a minimum with increasing single-walled carbon nanotube concentration," *Macromolecules*, vol. 42, no. 18, pp. 7091–7097, 2009.

[10] ——, "Polymer tracer diffusion exhibits a minimum in nanocomposites containing spherical nanoparticles," *Macromolecules*, vol. 44, no. 2, pp. 191–193, 2010.

[11] K. I. Winey and R. A. Vaia, "Polymer nanocomposites," *MRS bulletin*, vol. 32, no. 04, pp. 314–322, 2007.

[12] M. Muthukumar, "Entropic barrier model for polymer diffusion in concentrated polymer solutions and random media," *Journal of Non-Crystalline solids*, vol. 131, pp. 654–666, 1991.

[13] A. Baumgärtner, U. Ebert, and L. Schäfer, "Segment motion in the reptation model of polymer dynamics. ii. simulations," *Journal of Statistical Physics*, vol. 90, no. 5-6, pp. 1375–1400, 1998.

[14] K. Evans and S. Edwards, "Computer simulation of the dynamics of highly entangled polymers. part 1.equilibrium dynamics," *Journal of the Chemical Society, Faraday Transactions 2: Molecular and Chemical Physics*, vol. 77, no. 10, pp. 1891–1912, 1981.

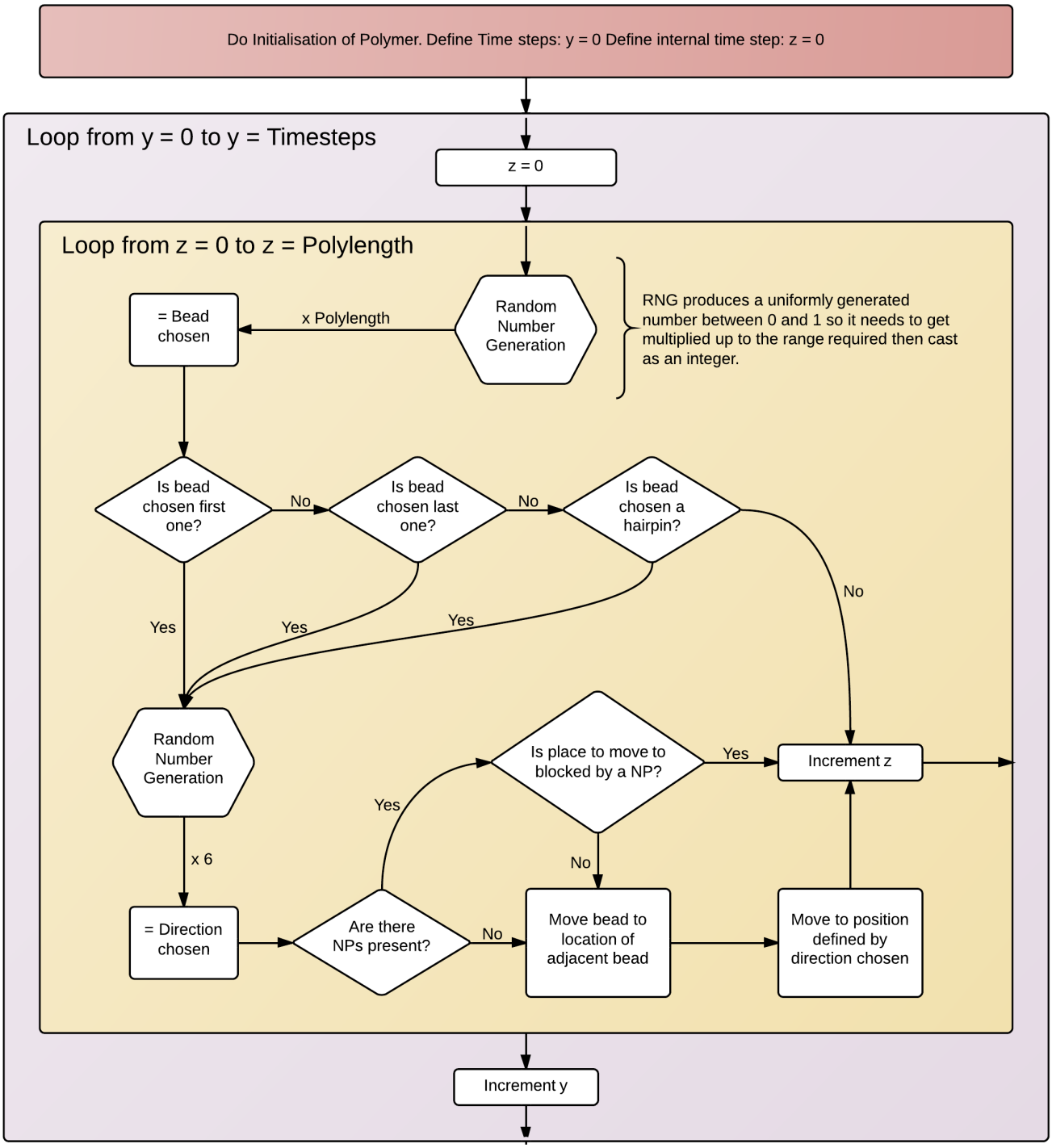
[15] C. Nvidia, "Programming guide," 2014.

[16] L. Howes and D. Thomas, "Efficient random number generation and application using cuda," *GPU gems*, vol. 3, pp. 805–830, 2007.

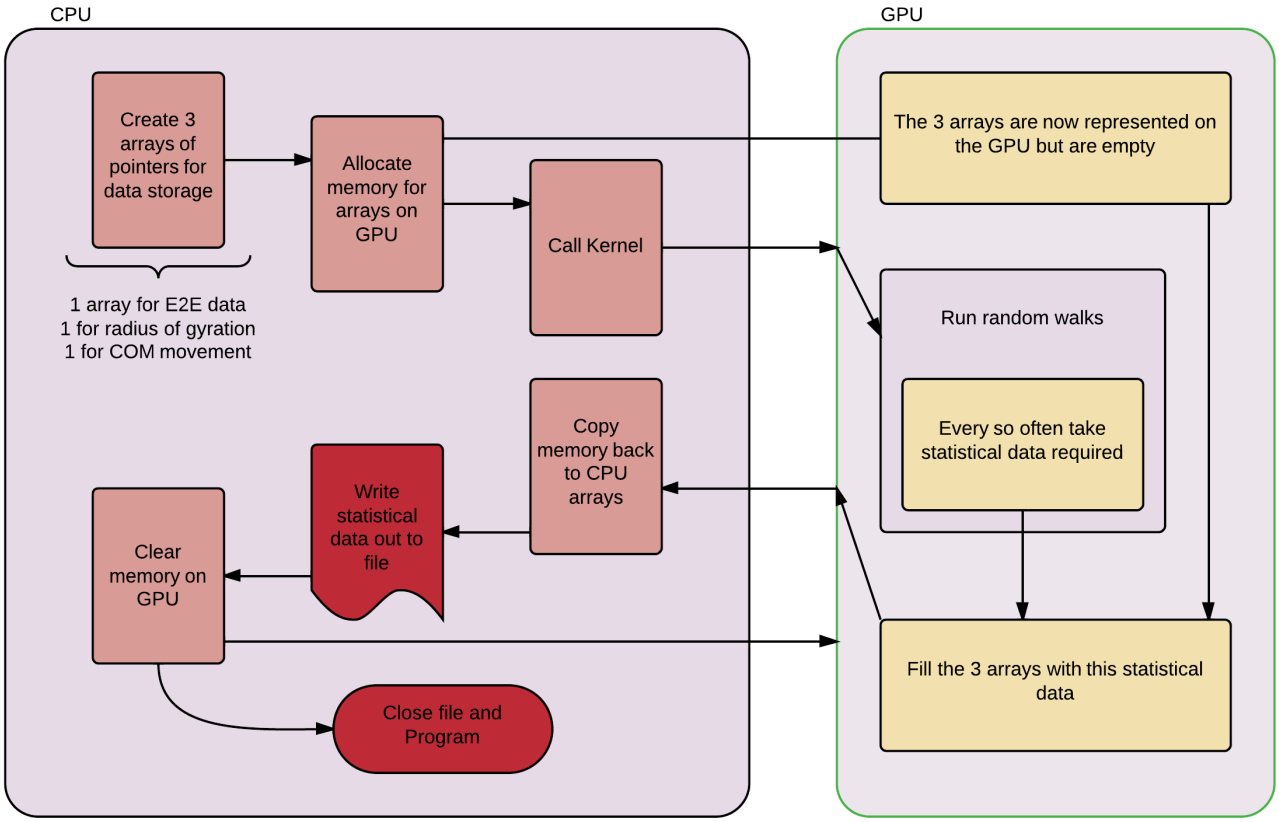
8 Appendix A - Simulation Logic

Theses flow charts produced by Glenn Jones helps demonstrate the functionality of the simulation:

The process in completing a random walk;



And the implementation of CUDA parallel processing;



9 Appendix B - Code

20/20 CUDA architecture simulating 896 polymers with N=100, 1 in 10 Pseudo random number nanoparticles:

```
#include <stdlib.h>
#include <iostream>
#include <cmath>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <fstream>
#include <cuda.h>
#include <map>
#include <cuda_runtime.h>
#include <device_launch_parameters.h>
#include <thrust/host_vector.h>
#include <thrust/device_vector.h>
#include <curand_kernel.h>
#include <curand.h>
using namespace std;
#define NoPOLY 896
#define TIMESTEPS 15000000
#define SUBDIFFUSE 2000000
#define DIFFUSE 2500000
#define DATAPOINTS (((((SUBDIFFUSE+(SUBDIFFUSE/10)) - (SUBDIFFUSE))/10) + (((DIFFUSE) - (SUBDIFFUSE+(SUBDIFFUSE/10)))/250) + (((DIFFUSE*5) - (DIFFUSE))/5000) + (((TIMESTEPS) - (DIFFUSE*5))/30000))+10)
#define POLYLENGTH 101
#define NANOSIZE 1
#define DENSITY 10

class statistics{
private:
  int n;
  float sum;
  float sumsq;
public:
  statistics();
  int getNumber() const;
  float getAverage() const;
  float getSqAverage() const;
  void add(float x);
};

__global__ void cudarandomwalk(float* placeend, float* d_endtoends, float* placebead, float* d_beadtomids, float* placerad, float* d_radofgys, size_t pitch, curandState *randStates, unsigned long seed) {
  int idx = blockDim.x * blockIdx.x + threadIdx.x;
  if (idx < NoPOLY)
  {
    curandState& randState = randStates[idx];
    curand_init(seed, idx, 0, &randState);
    int a=0;
    int datapointindex = 0;
    int currentnode = 0;
    int upnode;
    int downnode;
    int randomdir = 0;
    float endtoend;
    float beadtomid;
    float radofgy;
    int block;
    int resloop;
    int nloopx;
    int nloopy;
    int nloopz;
    float smidx = 0;
    float smidy = 0;
    float smidz = 0;
    float xSum = 0;
    float ySum = 0;
    float zSum = 0;
    float vrad = 0;
    int Gridx[POLYLENGTH];
    int Gridy[POLYLENGTH];
    int Gridz[POLYLENGTH];
    int Randomx[NANOSIZE];
    int Randomy[NANOSIZE];
    int Randomz[NANOSIZE];
    int b;
    int startx = (int)(curand_uniform(&randState)*1000) + 1000;
    int starty = (int)(curand_uniform(&randState)*1000) + 1000;
    int startz = (int)(curand_uniform(&randState)*1000) + 1000;
    for (nloopx = 0; nloopx < NANOSIZE; nloopx++)
    {
      Randomx[nloopx] = 0;
      Randomy[nloopx] = 0;
      Randomz[nloopx] = 0;
    }
    for (nloopx = 0; nloopx < NANOSIZE; nloopx++)
    {
      randomdir = (((int)(curand_uniform(&randState)*DENSITY)) %DENSITY);
      Randomx[nloopx] = randomdir;
      if (nloopx>0)
      {
        for (nloopy = 0; nloopy < (nloopx); nloopy++)
        {
          if (Randomx[nloopx] == Randomx[nloopy]) nloopx--;
        }
      }
    }
    for (nloopx = 0; nloopx < NANOSIZE; nloopx++)
    {
      randomdir = (((int)(curand_uniform(&randState)*DENSITY)) %DENSITY);
      Randomy[nloopx] = randomdir;

      if (nloopx>0)
      {
        for (nloopy = 0; nloopy < (nloopx); nloopy++)
        {
          if (Randomy[nloopx] == Randomy[nloopy]) nloopx--;
        }
      }
    }
    for (nloopx = 0; nloopx < NANOSIZE; nloopx++)
    {
      randomdir = (((int)(curand_uniform(&randState)*DENSITY)) %DENSITY);
      Randomz[nloopx] = randomdir;
      if (nloopx>0)
      {
        for (nloopy = 0; nloopy < (nloopx); nloopy++)
        {
          if (Randomz[nloopx] == Randomz[nloopy]) nloopx--;
        }
      }
    }
    for (a=0; a < POLYLENGTH; a++)
    {
      Gridx[a] = startx;
      Gridy[a] = starty;
      Gridz[a] = startz;
    }
    block=0;
    resloop=0;
    for(b=1; b < 2; b++)
    {
      block=0;
      for (nloopx = 0; nloopx < NANOSIZE; nloopx++)
      {
        for (nloopy = 0; nloopy < NANOSIZE; nloopy++)
        {
          for (nloopz = 0; nloopz < NANOSIZE; nloopz++)
          {
            if ((((((Gridx[a]+resloop)%DENSITY) == nloopx))
              && ((((Gridy[a]+resloop)%DENSITY) == nloopy))
              && ((((Gridz[a]+resloop)%DENSITY) == nloopz)))
                block = 1;
          }
        }
      }
      if (block == 1)
      {
        b--;
        resloop++;
      }
      if (block == 0)
      {
        for (a=0; a < POLYLENGTH; a++)
        {
          Gridx[a]= Gridx[a]+resloop;
          Gridy[a]= Gridy[a]+resloop;
          Gridz[a]= Gridz[a]+resloop;
        }
        b++;
      }
    }
    int incx[3] = { 1, 0, 0 };
    int incy[3] = { 0, 1, 0 };
    int incz[3] = { 0, 0, 1 };
    int inc2x[6] = { -1, 1, 0, 0, 0, 0 };
    int inc2y[6] = { 0, 0, -1, 1, 0, 0 };
    int inc2z[6] = { 0, 0, 0, 0, -1, 1 };
    for(a=0; a < POLYLENGTH; a++)
    {
      randomdir = (((int)(curand_uniform(&randState)*3))%3);
      block = 0;
      for (nloopx = 0; nloopx < NANOSIZE; nloopx++)
      {
        for (nloopy = 0; nloopy < NANOSIZE; nloopy++)
        {
          for (nloopz = 0; nloopz < NANOSIZE; nloopz++)
          {
            if ((((((Gridx[POLYLENGTH-1]+incx[randomdir])%DENSITY) == Randomx[nloopx])) && ((((Gridy[POLYLENGTH-1]+incy[randomdir])%DENSITY) == Randomy[nloopy])) && ((((Gridz[POLYLENGTH-1]+incz[randomdir])%DENSITY) == Randomz[nloopz]))) block = 1;
          }
        }
      }
      if (block == 0)
      {
        Gridx[POLYLENGTH-1]+=incx[randomdir];
        Gridy[POLYLENGTH-1]+=incy[randomdir];
        Gridz[POLYLENGTH-1]+=incz[randomdir];
        Gridx[a]=Gridx[POLYLENGTH-1];
        Gridy[a]=Gridy[POLYLENGTH-1];
        Gridz[a]=Gridz[POLYLENGTH-1];
      }
      if (block == 1) a--;
    }
    for (int y=0; y<=TIMESTEPS; y++)
    {

      for (int z=0; z<(POLYLENGTH); z++)
      {
        block = 0;
```

```
195         currentnode = (((int)(curand_uniform(&randState)*
                POLYLENGTH))%POLYLENGTH);
196         randomdir = (((int)(curand_uniform(&randState)*6))%6);
197         upnode=(currentnode+1);
198         downnode=(currentnode-1);
199         if (currentnode == 0)
200         {
201           for (nloopx = 0; nloopx < NANOSIZE; nloopx++)
202           {
203             for (nloopy = 0; nloopy < NANOSIZE; nloopy++)
204             {
205               for (nloopz = 0; nloopz < NANOSIZE; nloopz++)
206               {
207                 if ((((((Gridx[upnode]+inc2x[randomdir])%
                          DENSITY)) == Randomx[nloopx]) || ((((
                          Gridx[upnode]+inc2x[randomdir])%DENSITY))
                          == (Randomx[nloopx]-DENSITY))) &&
208                     (((((Gridy[upnode]+inc2y[randomdir])%DENSITY
                          )) == Randomy[nloopy]) || ((((Gridy[
                          upnode]+inc2y[randomdir])%DENSITY)) ==
                          (Randomy[nloopy]-DENSITY))) &&
209                     (((((Gridz[upnode]+inc2z[randomdir])%DENSITY
                          )) == Randomz[nloopz]) || ((((Gridz[
                          upnode]+inc2z[randomdir])%DENSITY)) ==
                          (Randomz[nloopz]-DENSITY)))) block =
                          1;
210               }
211             }
212           }
213
214           if (block == 0)
215           {
216             Gridx[currentnode] = Gridx[upnode];
217             Gridy[currentnode] = Gridy[upnode];
218             Gridz[currentnode] = Gridz[upnode];
219             Gridx[currentnode]+=inc2x[randomdir];
220             Gridy[currentnode]+=inc2y[randomdir];
221             Gridz[currentnode]+=inc2z[randomdir];
222           }
223
224         }
225
226         if (currentnode == (POLYLENGTH-1))
227         {
228           for (nloopx = 0; nloopx < NANOSIZE; nloopx++)
229           {
230             for (nloopy = 0; nloopy < NANOSIZE; nloopy++)
231             {
232               for (nloopz = 0; nloopz < NANOSIZE; nloopz++)
233               {
234                 if ((((((Gridx[downnode]+inc2x[randomdir])%
                          DENSITY)) == Randomx[nloopx]) || ((((
                          Gridx[downnode]+inc2x[randomdir])%DENSITY
                          )) == (Randomx[nloopx]-DENSITY))) &&
235                     (((((Gridy[downnode]+inc2y[randomdir])%
                          DENSITY)) == Randomy[nloopy]) || ((((
                          Gridy[downnode]+inc2y[randomdir])%
                          DENSITY)) == (Randomy[nloopy]-DENSITY)
                          )) &&
236                     (((((Gridz[downnode]+inc2z[randomdir])%
                          DENSITY)) == Randomz[nloopz]) || ((((
                          Gridz[downnode]+inc2z[randomdir])%
                          DENSITY)) == (Randomz[nloopz]-DENSITY)
                          ))) block = 1;
237               }
238             }
239           }
240
241           if (block == 0)
242           {
243             Gridx[currentnode] = Gridx[downnode];
244             Gridy[currentnode] = Gridy[downnode];
245             Gridz[currentnode] = Gridz[downnode];
246             Gridx[currentnode]+=inc2x[randomdir];
247             Gridy[currentnode]+=inc2y[randomdir];
248             Gridz[currentnode]+=inc2z[randomdir];
249           }
250         }
251
252         if ((0 < currentnode) && (currentnode < (POLYLENGTH-1)
              ))
253         {
254           if ((Gridx[downnode] == Gridx[upnode]) && (Gridy[
                downnode] == Gridy[upnode]) && (Gridz[downnode]
                == Gridz[upnode]) )
255           {
256             for (nloopx = 0; nloopx < NANOSIZE; nloopx++)
257             {
258               for (nloopy = 0; nloopy < NANOSIZE; nloopy++)
259               {
260                 for (nloopz = 0; nloopz < NANOSIZE; nloopz++)
261                 {
262                   if ((((((Gridx[upnode]+inc2x[randomdir])%
                            DENSITY)) == Randomx[nloopx]) || ((((
                            Gridx[upnode]+inc2x[randomdir])%DENSITY
                            )) == (Randomx[nloopx]-DENSITY))) &&
263                       (((((Gridy[upnode]+inc2y[randomdir])%
                            DENSITY)) == Randomy[nloopy]) || ((((
                            Gridy[upnode]+inc2y[randomdir])%
                            DENSITY)) == (Randomy[nloopy]-
                            DENSITY))) &&
264                       (((((Gridz[upnode]+inc2z[randomdir])%
                            DENSITY)) == Randomz[nloopz]) || ((((
                            Gridz[upnode]+inc2z[randomdir])%
                            DENSITY)) == (Randomz[nloopz]-
                            DENSITY)))) block = 1;
265                 }
266               }
267             }
268
269             if (block == 0)
270             {
271               Gridx[currentnode] = Gridx[upnode];
272               Gridy[currentnode] = Gridy[upnode];
273               Gridz[currentnode] = Gridz[upnode];
274               Gridx[currentnode]+=inc2x[randomdir];
275               Gridy[currentnode]+=inc2y[randomdir];
276               Gridz[currentnode]+=inc2z[randomdir];
277             }
278           }
279         }
280       }
281       if (y == SUBDIFFUSE)
282       {
283         for (a=0; a < POLYLENGTH; a++)
284         {
285           smidx = smidx + (float)Gridx[a];
286           smidy = smidy + (float)Gridy[a];
287           smidz = smidz + (float)Gridz[a];
288         }
289       }
290         if ((((y % 10) == 0) && (y >= (SUBDIFFUSE+10)) && (y
              <= (SUBDIFFUSE+(SUBDIFFUSE/10)))) ||
291           ((y % 250) == 0 && (y <= (DIFFUSE)) && (y > (
                SUBDIFFUSE+(SUBDIFFUSE/10)))) ||
292             ((y % 5000) == 0 && (y <= (DIFFUSE*5)) && (y >
                  DIFFUSE)) || ((y % 30000) == 0 && (y > (
                  DIFFUSE*5))))
293       {
294         endtoend = sqrt((float)((Gridx[POLYLENGTH - 1] - Gridx
                [0]) * (Gridx[POLYLENGTH - 1] - Gridx[0]) +
295           (Gridy[POLYLENGTH - 1] - Gridy[0]) * (Gridy[
                POLYLENGTH - 1] - Gridy[0]) +
296           (Gridz[POLYLENGTH - 1] - Gridz[0]) * (Gridz[
                POLYLENGTH - 1] - Gridz[0])));
297         xSum = 0;
298         ySum = 0;
299         zSum = 0;
300         vrad = 0;
301         for (a=0; a < POLYLENGTH; a++)
302         {
303           xSum = xSum + (float)Gridx[a];
304           ySum = ySum + (float)Gridy[a];
305           zSum = zSum + (float)Gridz[a];
306         }
307         xSum = (xSum)/float(POLYLENGTH);
308         ySum = (ySum)/float(POLYLENGTH);
309         zSum = (zSum)/float(POLYLENGTH);
310         for (a=0;a<POLYLENGTH;a++)
311         {
312           vrad = vrad + (((float(Gridx[a])) - xSum)*((float(
                Gridx[a])) - xSum)) +
313             (((float(Gridy[a])) - ySum)*((float(Gridy[a])) -
                  ySum)) +
314             (((float(Gridz[a])) - zSum)*((float(Gridz[a])) -
                  zSum));
315         }
316         radofgy = sqrt(vrad/(float(POLYLENGTH)));
317         xSum = xSum - (smidx/float(POLYLENGTH));
318         ySum = ySum - (smidy/float(POLYLENGTH));
319         zSum = zSum - (smidz/float(POLYLENGTH));
320         beadtomid = ((xSum*xSum)+(ySum*ySum)+(zSum*zSum));
321         float* datapointdata1 = (float*)(((char*)placeend) + (
                datapointindex*pitch));
322         float* datapointdata2 = (float*)(((char*)placebead) +
                (datapointindex*pitch));
323         float* datapointdata3 = (float*)(((char*)placerad) + (
                datapointindex*pitch));
324         datapointdata1[idx] = endtoend;
325         datapointdata2[idx] = beadtomid;
326         datapointdata3[idx] = radofgy;
327         d_endtoends[(datapointindex*NoPOLY) + idx] =
                datapointdata1[idx];
328         d_beadtomids[(datapointindex*NoPOLY) + idx] =
                datapointdata2[idx];
329         d_radofgys[(datapointindex*NoPOLY) + idx] =
                datapointdata3[idx];
330         datapointindex++;
331       }
332     }
333   }
334 }
335
336 statistics::statistics() {
337   n=0;
338   sum=0.0;
339   sumsq=0.0;
340 }
341
342 int statistics::getNumber() const{
343   return n;
344 }
345
346 float statistics::getAverage() const {
347   if(n==0) return 0.;
348   return sum/n;
349 }
350
351 float statistics::getSqAverage() const {
352   if(n==0) return -1;
353   return sumsq/n;
354 }
355
356 void statistics::add(float x) {
357   n++;
358   sum += x;
359   sumsq += x*x;
360 }
361
362 int main()
```

```
363 {
364   long startTime = clock();
365   long starttime2;
366   long finishtime2;
367   int y = 0;
368   int ig = 0;
369   int jg = 0;
370   float check1 = (((SUBDIFFUSE+(SUBDIFFUSE/10)) - (SUBDIFFUSE)
          )/10);
371   float check2 = (((DIFFUSE)  - (SUBDIFFUSE+(SUBDIFFUSE/10)))
          /250);
372   float check3 = (((DIFFUSE*5) - (DIFFUSE))/5000);
373   float check4 = (((TIMESTEPS) - (DIFFUSE*5))/30000);
374   float *placeend;
375   float *placebead;
376   float *placerad;
377   float *d_endtoends;
378   float *d_beadtomids;
379   float *d_radofgys;
380   float *h_endtoends = new float[DATAPOINTS*NoPOLY];
381   float *h_beadtomids = new float[DATAPOINTS*NoPOLY];
382   float *h_radofgys = new float[DATAPOINTS*NoPOLY];
383   curandState* randStates;
384   size_t pitch;
385   cudaMallocPitch(&placeend, &pitch, sizeof(float)*NoPOLY,
          DATAPOINTS);
386   cudaMalloc(&d_endtoends, sizeof(float)*NoPOLY*DATAPOINTS);
387   cudaMallocPitch(&placebead, &pitch, sizeof(float)*NoPOLY,
          DATAPOINTS);
388   cudaMalloc(&d_beadtomids, sizeof(float)*NoPOLY*DATAPOINTS);
389   cudaMallocPitch(&placerad, &pitch, sizeof(float)*NoPOLY,
          DATAPOINTS);
390   cudaMalloc(&d_radofgys, sizeof(float)*NoPOLY*DATAPOINTS);
391   cudaMalloc(&randStates, NoPOLY*sizeof(curandState));
392   cout << "Should be integers:" << check1 << "," << check2 <<
          "," << check3 << "," << check4 << endl;
393   ofstream outfile;
394   outfile.open ("THEONE_RALG_101_2in7.txt");
395   if (!outfile.is_open())
396   {
397     cout << "file not open" << endl;
398     return 42;
399   }
400   outfile << "TimeStep " << "E2EDistance " << "ErrorE2E " << "
          RadofGy " << "ErrorRadofgy " << "R^2 " << "ErrorR^2 "
          << "log10(TimeStep) " << "log10(R^2) " << "ErrorLog10(R
          ^2) " << endl;
401   starttime2 = clock();
402   cudarandomwalk<<<112, 8>>>(placeend, d_endtoends, placebead,
          d_beadtomids, placerad, d_radofgys, pitch, randStates,
          time(NULL));
403   cudaMemcpy(h_endtoends, d_endtoends, sizeof(float)*NoPOLY*
          DATAPOINTS, cudaMemcpyDeviceToHost);
404   cudaMemcpy(h_beadtomids, d_beadtomids, sizeof(float)*NoPOLY*
          DATAPOINTS, cudaMemcpyDeviceToHost);
405   cudaMemcpy(h_radofgys, d_radofgys, sizeof(float)*NoPOLY*
          DATAPOINTS, cudaMemcpyDeviceToHost);
406   finishtime2 = clock();
407   cout<<endl<<"Kernal Run time is "<<((finishtime2 -
          starttime2)/double(CLOCKS_PER_SEC))<<" seconds"<<endl<<
          endl;
408   for(y = (SUBDIFFUSE - 10) ; y < TIMESTEPS; y++)
409   {
410     statistics rsq;
411     statistics flength;
412     statistics rgst;
413     if ((((y % 10) == 0) && (y >= (SUBDIFFUSE+10)) && (y <= (
            SUBDIFFUSE+(SUBDIFFUSE/10)))) ||
414     ((y % 250) == 0 && (y <= (DIFFUSE)) && (y > (SUBDIFFUSE+(
            SUBDIFFUSE/10)))) ||
415     ((y % 5000) == 0 && (y <= (DIFFUSE*5)) && (y > DIFFUSE))
            || ((y % 30000) == 0 && (y > (DIFFUSE*5))))
416     {
417       for(ig = 0 ; ig < NoPOLY; ig++)
418       {
419         flength.add(h_endtoends[(jg*NoPOLY)+ig]);
420         rsq.add(h_beadtomids[(jg*NoPOLY)+ig]);
421         rgst.add(h_radofgys[(jg*NoPOLY)+ig]);
422       }
423       jg++;
424       outfile << (y-SUBDIFFUSE) << " " <<  flength.getAverage
            () << " "  << (sqrt(flength.getSqAverage() - (
            flength.getAverage()*flength.getAverage())/((float)
            NoPOLY)))
425         << " " <<  rgst.getAverage()  << " " << (sqrt(rgst.
              getSqAverage() - (rgst.getAverage()*rgst.
              getAverage())/((float)NoPOLY))) << " "
426         << rsq.getAverage() << " " << (sqrt(rsq.getSqAverage()
              - (rsq.getAverage()*rsq.getAverage())/((float)
            NoPOLY)))  << " " << log10((float)(y-SUBDIFFUSE))
              << " " << log10(rsq.getAverage())
427         << "  " << (sqrt(rsq.getSqAverage() - (rsq.getAverage
              ()*rsq.getAverage())/((float)NoPOLY)))/(rsq.
              getAverage()*2.302585) << endl;
428     }
429   }
430   cout << "-------------" << endl << "END OF RANDOMWALK" <<
          endl;
431   cudaFree(d_endtoends);
432   cudaFree(d_beadtomids);
433   cudaFree(d_radofgys);
434   cudaFree(placeend);
435   cudaFree(placebead);
436   cudaFree(placerad);
437   cudaFree(randStates);
438   cudaDeviceReset();
439   outfile.close();
440   long finishTime = clock();
441   cout<<endl<<"Program Run time is "<<((finishTime - startTime
          )/double(CLOCKS_PER_SEC))<<" seconds"<<endl<<endl;
442   return 0;
443 }
```