Nucleation of nanoparticles in a coarse grained fluid
using OpenCL

Sebastian Oude Voshaar (3141624)
Master's thesis Nanomaterials, 2014

Soft Condensed Matter group
Debye Institute
Utrecht University

ABSTRACT

In this thesis, the nucleation rate of almost hard spheres in a course-grained fluid is measured to study the effects of an explicit solvent on the nucleation rate. Previous measurements show a discrepancy between physical measurements and simulations, where the latter all used implicit solvents.

In this thesis, the fluid is approximated using Stochastic Rotation Dynamics (SRD), which natively contains Brownian dynamics as well as long range hydrodynamic forces between particles, and obeys the Navier-Stokes equations. The Poisseuille flow of the SRD fluid is examined, and found to match the Navier-Stokes equations. We also measured diffusion and velocity autocorrelation of nanoparticles in the fluid.

The nucleation rate matches experimental data, but is in significant disagreement with other simulations. The nucleation rate progresses as a less steep curve than implicit-solvent methods and soft-particle simulations suggest.

Since nucleation simulations are time-consuming, we used a graphics card in combination with OpenCL to speed up calculations. This sped up computation by roughly 40 times compared to standard processors. However, we found that specific attention must be paid to parallelisation issues and memory optimisations in this case.

Table of Contents

1 INTRODUCTION

Nucleation refers to the process where a new, more stable phase forms within an existing (metastable) phase. The classical example is the formation of ice in supercooled water. While nucleation is an important process in nature, studying this process is difficult because of its random nature, and the long waiting times associated with these rare events. Although the effects of a nucleation event are clear enough after it has happened, predicting where and when the nucleation itself occurs is impossible.
To study the phenomenon in more detail, scientists have turned to computer models to simulate nucleation. One of the classic model systems for comparing to experiments are hard colloidal particles as these can both be simulated as well as synthesized in a lab. Hard spheres, which are an example of hard colloidal particles, are a well-studied system, and their phase behaviour is well established[1]. However, in the case of nucleation, while many simulations[2][3] have been performed on hard spheres, they are not yet in agreement with experiments. In particular, the measurements on approximately hard colloidal spheres, such as sterically stabilized polymer spheres[4], produce a different nucleation rate at low concentrations than simulations suggest.

Different methods of simulation have been tested to resolve the nucleation rate, and currently all are in agreement with each other. However, all methods have in common that the solvent was included implicitly, e.g. the solvent particles themselves were not included in the simulations. In such methods, the effects of a solvent are represented by a formula which is applied to the particles.

In this thesis we investigate whether the use of explicit solvent particles, which in theory provides better hydrodynamics, results in nucleation rates that agree better with experiments. Since fully simulating a solvent during nucleation takes too much computation time, we choose to use an approximate method called Stochastic Rotation Dynamics (SRD). This method, developed in 1999 by Malevanets and Kapral[5]provides more accurate results than implicit-solvent methods while treating the interaction between the solvent particles in an approximate manner. Note that SRD itself does not completely remove the issue of long simulation times. To further shorten the long computation time inherent to crystallisation simulations, in this thesis a graphics card is used in combination with OpenCL. Solving the problems coming along with this new way of computation is a second focus of this thesis.

This thesis is divided in several parts; Chapter 2 will explore the theory behind molecular dynamics, and SRD in particular, because this thesis relies heavily on the latter. Classical nucleation theory and the methods to study nucleation are covered as well. Lastly, computation on graphics cards and OpenCL theory are addressed.
Chapter 3 is dedicated to the simulation details and its setup. The parameters used in this thesis are explained, and some insight is given to the problems encountered in getting OpenCL to work.
Chapter 4 discusses the results of the SRD tests, the nucleation detection and the nucleation rates, which are compared to other works in the field. It also shows the efficiency improvements by the graphics card, which are significant.
Chapter 5 contains the discussion of the results, and an outlook of further research possibilities.

2 THEORY

In this thesis, we examine the nucleation of nanoparticles that interact via an almost hard interaction potential. In contrast to previous studies, we will be simulating the nucleation of nanoparticles as well as the liquid medium in which they are dispersed. To do so, we use molecular dynamics to simulate the nanoparticles and model the liquid medium using SRD particles. Note that SRD reduces the amount of computation required for the solvent. We employ an approximate solvent since this is expected to be the heaviest computational task.

In this chapter we describe the potential we use to model the interaction between the nanoparticles. We also discuss the methods we use to simulate the dynamics. Additionally, we discuss the theory behind nucleation. Lastly, we explain how we put our simulation code onto a graphics card, and the differences between this and simulating on a normal CPU.

2.1 Model

2.1.1 Potentials

One of the most studied models to describe interactions between nanoparticles is the hard sphere potential. This potential is given by

$$\beta E = \infty \qquad r < \sigma \quad , \tag{2.1}$$

$$\beta E = 0 \qquad r > \sigma \quad , \tag{2.2}$$

where r is the distance between the particles, σ is the diameter of the particles and β is 1/kT, where k is Boltzmanns constant, and T the temperature. This potential describes the interactions in colloidal systems such as those studied by Harland & Van Meegen[4], and significant effort has been put in characterizing the associated nucleation rates. Using typical molecular dynamics simulations with hard particles is not possible, because at the discontinuity the force would become infinite. Thus, instead of modelling hard spheres, in this thesis we will use the Weeks Chandler Anderson[6] (WCA) potential because it offers a soft repulsion that is continuously and rapidly decaying to 0. Note that the WCA potential is based on the Lennard Jones potential, and truncated at the potential minimum. The exact pair potential βE for particles with diameter σ at distance r is given by:

$$\beta E = 4\beta\epsilon\left(\left(\frac{\sigma}{r}\right)^{2\alpha} - \left(\frac{\sigma}{r}\right)^{\alpha} + \frac{1}{4}\right) \qquad r < 2^{\frac{1}{\alpha}}\sigma \quad , \tag{2.3}$$

$$\beta E = 0 \qquad r > 2^{\frac{1}{\alpha}}\sigma \quad , \tag{2.4}$$

where ε is the interaction strength, measured in kT. The parameter α determines the sharpness of the interaction, with $\alpha \to \infty$ corresponding to a hard particle.

In this thesis, we use 2 different values of α, which are notated as WCA α - 2 α. A WCA24-48 potential was used to test the SRD fluid by comparing the results to the work of Padding and Louis[7] and Malevanets and Kapral. The WCA6-12 was used for the measurements on nucleation. Figure 1 shows a plot of both potentials, with the WCA24-48 having a βε of 1, and the WCA6-12 having a βε of 40. The LJ6-12 potential is given as reference, with a βε of 40 as well.

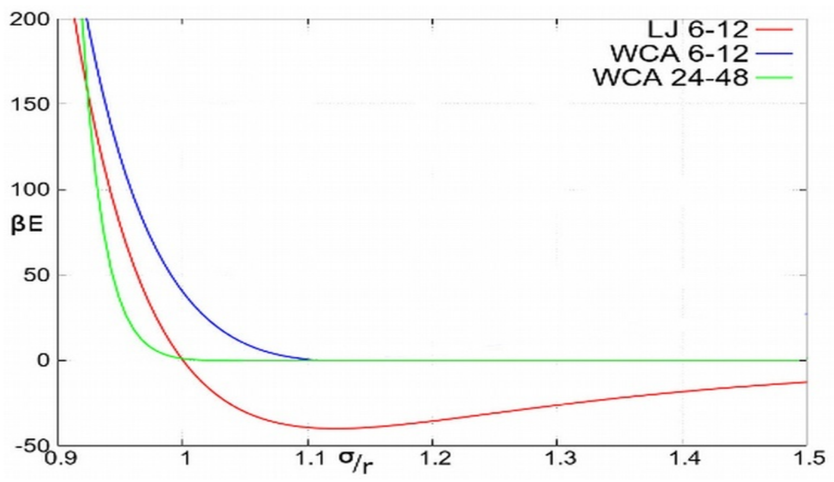

Figure 1: A comparison of the standard Lennard Jones potential with the Weeks-Chandler-Anderson (WCA) potential with α = 6 and α = 24. Both 6-12 potentials have a βε of 40, while the WCA 24-48 has a βε of 1.

2.1.2 Molecular Dynamics

One way to study phase transitions in many-body systems is to use Molecular Dynamics (MD) simulations. MD uses Newtons equations of motion to describe movement of particles over time. The equations of motion for each particle are determined by summing all forces acting on the particle using Newtons second law:

$$\vec{F}_i = m a = \vec{F}_E + \sum_{i \neq j} \vec{F}_{ij} \quad , \tag{2.5}$$

where F_i is the total force acting on particle i, m is the mass of the particle, a is the acceleration, F_{ij} is the force between particles i and j arising due to the interaction between particles i and j, and F_E is the force due to any external fields.
In order to simulate the motion of such particles, one needs to integrate equation 1.5. Many algorithms have been suggested to integrate the equation of motion, such as the leapfrog algorithm, Beemans algorithm or various types of Verlet integration. In this thesis, we use the Velocity Verlet algorithm, which is quite stable (the error scales with Δt^2) and which treats all particles at the same moment in time, and is thus inherently parallel. In Velocity Verlet, the position of a particle at time t + Δt is given by:

$$x(t+\Delta t) = x(t) + v(t)\Delta t + \frac{1}{2} a(t) \Delta t^2 \quad , \tag{2.6}$$

and the velocity is given by:

$$v(t+\Delta t) = v(t) + \frac{a(t)+a(t+\Delta t)}{2} \Delta t \quad , \tag{2.7}$$

where *x(t)*, *v(t)* and *a(t)* are the position, velocity and acceleration respectively of a particle at time t.
Equations 1.6 and 1.7 give the location and velocity of a particle after a timestep Δt based on its current velocity and acceleration, as well as the acceleration after one Δt. This calculation is generally split in 2 steps, in which first the old, and then the new acceleration are added to the velocity, whereas the location is updated only at the first step.

2.1.3 Solvents and Stochastic Rotation Dynamics

Since we focus on nucleation of nanoparticles in a liquid, this means that the liquid itself has to be simulated as well. A fluid medium expands the number of interactions considerably, because both the nanoparticles and the particles that make up the liquid have to be considered. Additionally, the density of a liquid is high enough to produce a domino effect among the liquid particles, a moving nanoparticle can influence many others around it, not just its nearest neighbours. These so-called long-range interactions (LRIs) may play a significant role in nucleation; the method chosen to simulate the liquid should resolve those forces accurately.

When doing molecular simulations in a solvent, the amount of computation time required to simulate this solvent is an important factor in deciding which method to use. One can either simulate the particles explicitly, which can take over 90% of the total time, or treat the solvent statistically, in which case the solvent is treated as a set of effective forces applied to the nanoparticles. The dynamics, however, will be less accurately modelled if no explicit particles are present. One example of such an approximation is Brownian Dynamics[8]. In Brownian Dynamics the motion of a particle is considered to be overdamped due to the friction with the solvent. As a result the equations of motion of a particle are given by

$$\frac{dx}{dt} = \frac{-\nabla v(x)}{\gamma m} + S(t) \quad , \tag{2.8}$$

where γ is the friction coefficient, m the mass of the particle, and $S(t)=\sqrt{(2D)}R(t)$ is a randomisation factor composed of the Einstein diffusion coefficient D and random displacement vector R(t), which is modelled by a Gaussian function with zero mean and unit variance.

Brownian Dynamics, although much faster than treating the particles explicitly, is an approximation that neglects flow and momentum transports through the fluid[7], which are generally called Long Range Interactions (LRIs). Attempts at adding these LRIs are computationally intensive ($O(N^3)$ in the implementation of Ermak & McCammon[8]), thus reducing usability. A solution to this is to approximate the LRIs, as proposed by Geyer & Winter[9]. However, such implicit-solvent methods work best for large and rigid nanoparticles, that only have weak dependence on the exact solvent-particle properties.

When such a method is insufficient (for example when studying crystallisation), one has to explicitly include the solvent particles, preferably in a computationally light way. For example, one can replace Newtons equations and the required force computation by the Boltzmann equation, which uses a probability distribution function. A method often used is Lattice Boltzmann[10], which limits the solutions of the Boltzmann equation to a grid, on which the particles are allowed to move. If particles are found to collide, the commonly used solution (the BGK operator) sets their new velocities and directions according to a Maxwell distribution, instead of calculating the exact values for these. This will make movements and collision detection easy, and will provide proper Navier-Stokes fluid behaviour, although the Brownian motion is still introduced per particle and in a random fashion. However, the drawback is the lattice spacing, which has to be coarse enough not to introduce too heavy a computation penalty, and still provide the required resolution and particle flexibility.

In most methods which use discrete solvents, the fluid-fluid interactions play a significant role. Since the ratio of fluids to larger particles is generally high (1000:1 is no exception), these determine the efficiency of the method to a large extent. To simplify the interactions, but avoid using a grid, an alternative method called Stochastic Rotation Dynamics (SRD) has been suggested by Malevanets & Kapral in 1999[5], and this method is used in this thesis.

Stochastic Rotation Dynamics (SRD) focuses on simplifying solvent-solvent interactions, and was invented in 1999 by Malevanets and Kapral[5]. It does so by replacing the actual liquid particles by pseudoparticles, of which there are fewer, and replacing the interactions by averaging velocities over nearby particles.

In an SRD simulation, the space is divided into cells of a specific size, and filled with pseudoparticles. All interactions between these liquid pseudoparticles are replaced by averaging and rotating the particles per cell. In order to retain momentum of the liquid (per cell, not per liquid particle), rotations have to be performed around the centre of mass of a cell. To ensure molecular chaos (sufficient spread of energy distribution and direction) cells are relatively small in size (on average 5 particles per cell) and the cell boundaries are shifted per iteration, so averaging happens over different sets of particles every time. The liquid still has proper pressure due to this averaging, as long as nanoparticles are of sufficient size (roughly 4 cells diameter minimum)[7].
Rotations of a velocity v_i are done according to:

$$v_i = v_{cm} + R(v_i - v_{cm}) \quad , \tag{2.9}$$

where R is a rotation matrix, rotating 90 degrees over a random axis (which varies per iteration and cell) around the velocity centre of mass of the cell, denoted by v_{cm}. This is an efficient way of spreading momentum between particles, while keeping sufficient Brownian motion in the simulation on a local scale. Because the orientation per cell is random, the Brownian motion evens out on a short scale. However, the general velocity of the liquid particles is maintained because the average velocity is kept intact.

The interactions with other particles in the system are done classically via Newtons equations, where SRD pseudoparticles have a defined weight (m_0) and diameter, which is generally 0. The interaction between the SRD particles and the other particles present in the system depends on the system in question. In our case a standard WCA potential was chosen, with the liquid acting as point particle (no diameter).

As SRD is used to simulate a fluid in this thesis, one has to ensure that the solvent particles are actually resembling a fluid. Padding and Louis[7] found that the properties of the SRD fluid such as compressibility and force transportation depend heavily on the simulation parameters.

The main parameter is the free path length λ, which can be expressed as

$$\lambda = \langle v \rangle\, t_{collision}/t_0\,, \tag{2.10}$$

where <v> is the average velocity and $t_{collision}$ is the average time a particle travels between 2 collisions and corresponds to the time between 2 SRD rotations. t_0, is the unit of time in the simulations. The longer a particle can move uninterrupted, the more energy will be transferred by diffusion, instead of by internal transfers. The rate between these energy transfers is usually expressed in the Schmidt number Sc, which is given by viscosity over the diffusion. Gasses tend to have an Sc of around 1, whereas liquids, where energy is transferred efficiently between many particles (due to the frequent collisions) have a high Sc. For the SRD liquid, the Schmidt number scales roughly with $1/\lambda^2$. Thus, to simulate a fluid, one will have to keep the free path length well below 1. Throughout this thesis, a λ of 0.1 will be used, which is sufficient for fluidlike behaviour.

Another important parameter is γ, the amount of particles per cell. This value must be high enough to ensures proper fluid density and viscosity, but low enough that it does not impact computation too much.
Padding and Louis show that viscosity only weakly depends on γ, but this is only valid for liquids. The SRD method decouples γ and λ, which in a gas are coupled ($\lambda = 1/\gamma$), and this allows some variations in viscosity if necessary. Since more particles per cell requires more computation, γ must be balanced between accuracy of the SRD simulation and computational speed.
Malevanets & Kapral[5] in their first SRD simulations used $\gamma = 10$ for their 2D simulations. Padding & Louis, which is used as reference in this thesis, preferred $\gamma=5$ for the 3D case, which is significantly lower, but still manages to keep the viscosity high enough to simulate a liquid. Cells are 1 length unit a_0 along all sides, which gives proper viscosity behaviour. When kT has been set to 1, a solvent particle with a mass of 1 m_f will roughly advance 1 a_0 in 1 t_0.

When interacting with other particles, a fluid may develop some undesired side-effects, such as depletion forces. The large diameter of a nanoparticle, as well as the interaction potential of the (larger) nanoparticles in a system, can have a significant effect on the behaviour of the particles. If two large bodies overlap, solvent particles can no longer enter the space between them and push them apart, thus only the outside pressure on the nanoparticles is left, pushing the particles together.

For a dilute solution such forces are usually around 1 kT, but when considering densely packed volumes forces may exceed this by many times. For a detailed explanation see Asakura and Oosawa[11].

Especially if the solvent particles cannot get close to the nanoparticles (because of their radius, or because of a different interaction potential between nanoparticles and solvent particles) these effects occur.

In our SRD simulation, we mostly negate the effects by using a slip boundary condition as well as a zero radius for the solvent particles, which according to Padding & Louis[7] effectively counters the effects of depletion.

2.2 Nucleation

Nucleation refers to the very first stages of forming a crystal, where small collections of particles crystallize and from a nucleus. When these nuclei become sufficiently large, they become stable and grow by simply adding particles from their surroundings. However, the initial stage is not that straightforward.
How exactly a nucleus is formed is not yet completely understood[12], however, classical nucleation theory (CNT) does explain most of the mechanisms behind it.

Note that nucleation is generally a rare event which means that it can take a long time for a nucleation event to happen. The nucleation rate k is given by

$$k = \frac{1}{\langle t \rangle V} \quad , \tag{2.11}$$

where <t> is the average time until a nucleation event, and V is the volume of the system.
This low nucleation rate is particularly a problem for computer simulations as the time that can be simulated is fairly limited. As a result, one may need to use more advanced simulation techniques in order to study the nucleation process, such as Umbrella Sampling or Forward Flux Sampling. However, in this thesis, we stick to nucleation processes which are sufficiently fast that we can observe them using brute force methods in combination with graphics cards.
In this section we introduce classical nucleation theory as well as some of the methods for studying rare events.

2.2.1 Classical Nucleation Theory

According to classical nucleation theory, the free energy difference associated with forming a nucleus in a supersaturated solvent, ΔG, is the sum of the surface free energy, ΔG_s and the free energy difference between the bulk crystal and supersaturated liquid, ΔG_v:

$$\Delta G = \Delta G_s + \Delta G_v = \gamma s + (\mu - \mu_E) N \quad , \tag{2.12}$$

where γ is the surface tension, s is the area of the surface of a nucleus containing N particles, μ is the chemical potential of the bulk crystal phase and μ_E is the chemical potential of the supersaturated liquid. Note that ΔG_s is always positive and ΔG_v is always negative.
For a spherical nucleus, which has the lowest surface area per volume, in an supersaturated medium, this can be rewritten as

$$\Delta G(r) = 4\pi r^2 \gamma + \frac{4}{3}\pi r^3 \Delta\mu\rho \quad , \tag{2.13}$$

with r the radius of a nucleus, ρ is the density of the bulk solid, and $\Delta\mu$ the potential difference as a result of the oversaturation. The function $\Delta G(r)$ is often referred to as the free-energy barrier. In Figure 2 we plot an example of a free-energy barrier. This free energy barrier has a maximum of ΔG_c at the critical radius R_c, where

$$\Delta G_c = \frac{16\pi\gamma^3}{3(\mu - \mu_E)^2 \rho^2} \quad , \tag{2.14}$$

Using CNT, the nucleation rate can be related to the free-energy barrier by

$$k = A \exp\left(\frac{-\Delta G_c}{kT}\right) \quad , \tag{2.15}$$

where A is a kinetic prefactor. As can be seen, the nucleation rate k is exponentially dependent on the height of the free-energy barrier which depend on the surface tension and supersaturation. For an elaborate explanation see Auer and Frenkel[13].

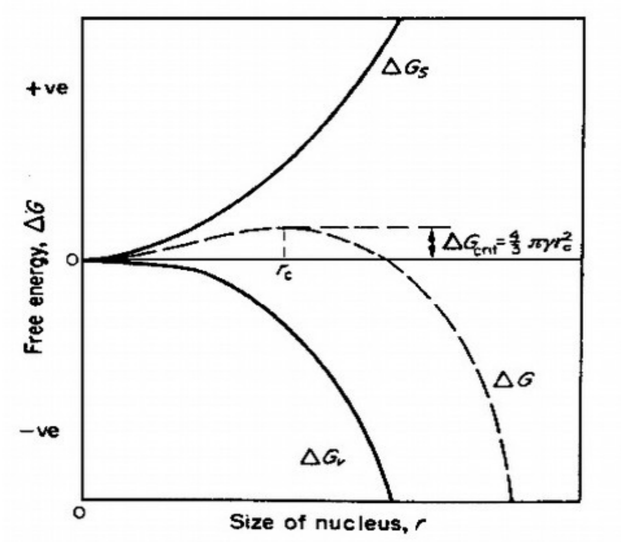

Figure 2: the relation between nucleus size and its free energy ΔG, with both components ΔGs (surface free energy) and ΔGv (volume free energy). The energy barrier ΔGcrit is at the maximum of ΔG. From Mullins[12]

2.2.2 Methods to compute crystallisation

Since nucleation is a very rare event at the densities of interest, several methods have been developed to increase the likelihood of the event appearing by pushing the system in the desired direction. Two of these methods are Umbrella Sampling[14] and Forward Flux Sampling[15]. Both of these methods require a measure of the amount of crystallinity in the system. To this end, an order parameter is introduced, which varies between 0 and 1 for the totally random and a crystalline state respectively.

Umbrella Sampling (US) is a method in which a biasing potential is used to flatten the potential landscape, so the transition of interest will occur much more often, and can thus be studied better. Afterwards one has to debias the results, correcting for the applied biasing.
However, this biasing potential depends on the order parameter, and thus a properly chosen order parameter is paramount in obtaining the correct results. Reducing the often multi-dimensional reaction landscape to one or sometimes 2 order parameters is a difficult task, and if done incorrectly, this can influence results.

Another method is Forward Flux Sampling (FFS)[15], where the transition of interest is divided in multiple separate steps. The nucleation process is then promoted by keeping the system in its successive intermediate states of a transition, which are defined by the so-called reaction coordinates. For hard spheres, the reaction coordinate is typically the same as the order parameter used in US. From a given position several attempts are launched to reach the next intermediate state, and the chance of achieving this is calculated. In the end the chances of all steps are then multiplied, which gives the chance of the complete event happening.
Evaluating these steps requires statistics, and once again a bad choice of reaction coordinates will lead to either very slow sampling or even incorrect results when an insufficient number of samples is taken.

Both methods are highly parallelisable (from an intermediate state a number of attempts can be executed at the same time), and since the size of the intermediate steps in FFS can generally be reduced, the chance of a simulated success becomes very close to one. For US one can increase the biasing potential, which also increases the chance. However, for both methods many runs are necessary to obtain a good estimate of the nucleation rate.

Since the methods presented above still contain a number of drawbacks, in particular their sensitivity to choosing a good order parameter, their use should be avoided unless there are no other options. In this thesis we therefore opted to use a plain simulation, and speed this up as far as reasonably possible, using a graphics card, as is detailed in the next section.

2.3 Graphics card computation and OpenCL

Direct simulation of nucleation tends to take a very long time, especially in the presence of an explicit solvent. As our attempts to reduce computation time using straightforward parallelisation proved insufficient, we looked at graphics cards, which sacrifice flexibility for raw processing power. While these pieces of hardware are abundant, and powerful, their use is still limited, mostly because of the specific demands on software that these potentially efficient cards have.

2.3.1 Parallel computing and GPUs

Since the advent of computers, computation has continually become faster. In the beginning this meant increasing clock speeds, so the single central processing unit (CPU) could execute more commands in the same time. However, this only provided temporary relief; around clock frequencies of 4 GHz a barrier was hit. Efficiency requirements and electronic path length forced engineers to find other solutions, with one solution being multiple processing units working together. This, however, brought its own share of problems with it, as the program had to be told which parts it could spread over the different units (cores), and which parts were best done on just a single core, leaving the others either idle, or doing something else in the meantime.

One highly parallel type of hardware is the graphics processor, and since 2004 it has also become possible to let the graphics processor (GPU) do some computational work[16], and feed the results back to the main processor. Because a modern GPU is a specialized piece of hardware, having up to several thousand highly specialised processing units, it cannot execute a complete program, or even understand it. It once was designed exclusively to process images, and put these on the attached screen. Around 1995 these tasks involved basic computations on 3D images (tessellation and afterwards pixel shading, which are geometric operations), but although basic, the operations came in large quantities.

Since a graphics card only understands very simple commands, a different programming language is used. In this thesis we will use the OpenCL framework, which is an extension of the C programming language. The main alternative is CUDA, which is proprietary, and only runs on one brand of hardware. As such we chose OpenCL. Several other languages and modules have been developed, notably CAL and Brook+[17] by AMD, but these are now deprecated in favour of OpenCL, which all major players support.

2.3.2 Stream computing and kernels

The basic idea of OpenCL is distributing the work into so-called kernels, small pieces of code that can be executed in parallel, on a specific device. These kernels can be compared to subroutines or methods in other languages. The main program starts and controls a multitude of kernels, feeding them data and processing the output. A kernel is only compiled at runtime[18], contrasting to for example C code which is compiled separately, and is then run.

Instances of these kernels are launched when needed, using a concept called stream processing, where one set of instructions is applied to a stream of data, instead of explicitly applying the instructions to separate pieces of data, as is more common on CPUs. This allows more flexibility in kernel instance numbers, and far less overhead for the thousands of kernels that can be active at the same time.
On the other hand, this new approach requires a change from the programmer, who will have to create kernels such that they can run with minimal dependence on the main thread, and independent of each other or data provided by other kernels. Global control is exercised by waiting until all kernels of a certain type are finished before launching the next type.

In many cases however communication is required between kernels, one of these cases being our simulation. The solution then is to launch kernels together in a so called 'work group'. These work groups are sets of identical kernels which are all executed parallel, and share a certain amount of memory. This shared memory can be used to exchange parameters. These work groups can also use barriers, which ensure that all kernels are synchronised. A barrier requires all kernels to reach it before allowing a kernel to proceed past the barrier.
Work groups are limited in size to one compute unit, which can handle between 32 and 256 kernels in parallel, depending on the hardware used. Global barriers are not yet supported in OpenCL.
Several of these work groups can be running in parallel, hardware permitting, but no direct communication is possible between them.

3 METHOD

To actually run the simulations, one needs a specific setup, as well as a method to detect crystals in a simulation.
Section 3.1 explains the simulation parameters and the setup required to produce nucleations.
Section 3.2 focusses on the adaptations we made to get the simulation working on a graphics card, among which are the multithreaded cell list and a custom random number generator. It also explains how efficiency is tested, and the pitfalls that come with these measurements.
Section 3.3 details the measuring of nucleation and how to distinguish between crystalline and liquid particles.

3.1 Simulation Details

In this chapter we describe the setup for the nucleation simulations done in this thesis. In our simulations we use a thermostat to keep the temperature of the system fixed throughout the simulation, and a combination of cell lists and Verlet lists to speed up the calculation of the forces. These will be described in more detail in the following 3 subsections.

3.1.1 Box setup

For each simulation a box is filled randomly with a specific number of particles (varying between 3000 and 3300) with a diameter σ of 4 a_0, where a_0 is the size of an SRD cell, and the used length scale in this thesis. The box is then compressed to the starting volume of 64^3 a_0^3 without liquid. Afterwards the liquid particles are added randomly, with the constraint that no overlap between nanoparticle and solvent occurs. The number of liquid particles added is 5 per cell, minus the amount of space taken by the nanoparticles. This corresponds to a correction of 167 solvent particles per nanoparticle. The interaction strength is increased from 0 to the desired level ($\beta\varepsilon = 40$ for both nano-nano and nano-solvent interactions) in the first steps of the simulation with 0.05 per t_0. The simulation box uses periodical boundary conditions, which means that particles leaving the box on one side are returned on the opposite side.

Basic units		
Energy	kT	1
Length of an SRD cell	a_0	1
Time = a_0 / sqrt(m_f/kT)	t_0	1
Mass	m_{fluid}	1
Simulation parameters		
Potential strength	$\beta\ \varepsilon$	40
Packing fraction	φ	0.39 – 0.41
Free path length	λ	0.1 a_0
Liquid particles per a_0^3	γ	5
Rotation matrix SRD	R	90°
Timestep SRD	$\Delta t(SRD)$	0.1 t_0
Timestep MD	$\Delta t(MD)$	0.0125 t_0
Mass of a colloid	m_{nano}	167 m_{fluid}
Radius of a colloid	r_{nano}	2 a_0

Table 1: Simulation parameters and used values

As explained in section 2.1.3 an SRD simulation involves both MD steps as well as solvent rotations. In our simulation a single time unit t_0 corresponds to 80 MD steps and 10 SRD rotations. Note that we investigated the number of MD steps per time unit and found that fewer MD steps led to unreliable results, as characterised by rapidly increasing system energies. The used parameters can be found in table 1.

3.1.2 Thermostat

Increasing the interaction strength as described in the box setup means that the total energy of the simulation is also increasing. The total energy, which is correlated to the temperature of the particles, is supposed to remain constant, and to achieve this, we use a thermostat. This thermostat follows Padding&Louis[7], where the temperature is measured every SRD iteration (thus 10 times per t_0). The temperature of the liquid, T_{act}, is taken from an average over all cells with 2 or more liquid particles in it.

$$k_b T_{act} = \frac{m_{fluid}}{N} \sum_c \sum_{i \in c} (v_i - v_{cm,c})^2 \quad , \tag{3.1}$$

where m_{fluid} is the mass of an SRD particle, N is the total number of liquid particles that are in a cell occupied by at least 2 liquid particles, c is the number of those cells, $v_{cm,c}$ the centre of velocity mass of cell c, and v_i - $v_{cm,c}$ is the relative velocity of particle I with respect to $V_{cm,c}$. After computing the temperature, the thermostating process scales all relative velocities of the particles by $\sqrt{(T/T_{act})}$.
Additionally, because of the increasing interaction strengths during start-up, in the initial phases the temperature is set to the original T once it drifts more than 2% from the desired value by scaling the velocities directly. This generally happens only a few times, and only in initial stages.

3.1.3 Cell lists and Verlet Lists

Computing the forces at each time step involves evaluating the distances between all particles, and computing the resulting potential. This is the most time-consuming part of the Velocity Verlet algorithm, and has to be done every iteration. However, if a particle only has interactions with its nearest neighbours, which are never further than a few units of length away, not all pairs have to be checked, and one can divide the space into separate cells, and only compare the particles in cells that are in each other's vicinity. A list of all particles in a certain cell, called a cell list, can reduce the required computation time by a large amount. Creating the cell list may take time, but this is far less than would otherwise be spent on computing unnecessary interactions. For larger simulations, such as nucleation simulations, which can have thousands of particles, a cell list or an alternative for it becomes a necessity.

The main alternative for a cell list is a Verlet list[19], where a list is kept per particle, detailing which particles are in its vicinity. This type of list can be constructed easily, but requires large quantities of memory and computer time (updating the list is an $O(N^2)$ process). Its memory constraints are especially significant if long interaction ranges or high densities are used, since a complete list has to be kept per particle.

In some cases, the particle list is then rearranged [20] to optimise data transport, but this in itself takes a lot of overhead, and resorting is again an $O(N^2)$ process, with incoherent data movement. We have chosen against resorting mainly because the incoherence of its memory access pattern wreaks havoc on the efficiency of a GPU. The effect of resorting is also limited, since particle movement can be done in a random order, and if one pre-loads the particles per cell and then performs all operations at once, the penalty for random memory access is only applied once. An option suggested by Anderson[20] would be Hilbert sorting[21], but this still requires a full resort every few iterations.

3.2 Nucleation Rates and detection

The nucleation rates from the WCA potential with $\beta\varepsilon = 40$ have previously been determined by Filion & Ni[2] as well as Kawasaki & Tanaka[3]. From this work we can identify a density range where nucleation is expected to occur on computationally accessible time scales. Based on these results we perform simulations at densities ranging from $\rho\sigma^3 = 0.750$ to $\rho\sigma^3 = 0.785$. This corresponds to a WCA packing fraction of $\varphi = 0.39$ to $\varphi = 0.41$ or a hard sphere packing fraction of $\varphi = 0.52$ to $\varphi = 0.54$. In this region multiple nucleations are expected to be highly unlikely so that we can limit ourselves to searching for the largest nucleus in the simulation box. The number of particles is chosen large enough so that the nucleation will not lower the particle pressure enough to prevent further crystallisation (Kawasaki & Tanaka found no negative effects at 4000 particles).
To measure the nucleation rates we performed simulations lasting 10k to 50k t_0 and recorded the particle positions every 10 t_0. In each snapshot we determined the size of the largest cluster as described below. When the size of this cluster surpasses 75 particles we consider the system nucleated, and when the cluster grows to 100 particles or more it is a stable cluster.

Detection of those nuclei is done via the bond order parameter, which in this thesis is calculated following Ten Wolde[22]. The bond order measures how much the particle resembles part of a solid, or part of a liquid. To qualify as a solid, a particle needs enough nearest neighbours that share the same environment.
The surroundings of a particle are evaluated up to a distance of 1.4 σ, which is slightly below the first minimum of an FCC crystal, thus only the first layer of particles is counted, and loose structures are adequately ignored.

To define a solid-like particle, the local orientational parameter is determined per particle, which is given by

$$q_l^m(i) = \frac{1}{N_b(i)} \sum_{j=0}^{N_b(i)} Y_l^m(\hat{r}_{ij}) \quad , \tag{3.2}$$

where N_b is the number of neighbours for a particle i, Y_l^m are the spherical harmonics, and $\hat{r}_{ij}$ is a unit vector pointing in the direction of particle j. We limit ourselves to the case of $l = 6$, which allows for the detection of the crystals expected in hard shperes, namely fcc and hcp[23]. We can then compute how much the particles are oriented alike by computing the q_6q_6-order as:

$$q_6q_6(ij) = \frac{1}{\widetilde{N}} \sum_{m=-6}^{6} q_6^m(i) \cdot q^{*}{}_6^m(j) \quad with \ \widetilde{N} = \left(\sum_{m=-6}^{6} \left| q_6^m(i) \right|^2 \right)^{\frac{1}{2}} \left(\sum_{m=-6}^{6} \left| q_6^m(j) \right|^2 \right)^{\frac{1}{2}} \quad , \tag{3.3}$$

where $q^*_6(j)$ is the conjugate of $q_6(j)$ and $\widetilde{N}$ is the normalisation constant. This q_6q_6 parameter indicates how much the particles are 'connected', i.e. share the same surroundings. If both particles are solid-like, the q_6q_6 will be close to 1, and if both are not related, it will be 0. Our cut-off for a bond is 0.7, which requires particles to align reasonably well. If this cut-off is reached, we call it a bond. For each particle the number of bonds is then determined, and if this reaches the nbond-cut-off (Nc), the particle is called crystalline. This Nc-parameter in our case is set to 8 (out of 12). This value is rather high, but we are not interested in the exact location of the surface (where the number of bonds is lower, around 5-6).

Afterwards we count how many of these crystalline particles form a cluster, with the limit that one particle can only be a member of one cluster at a time, and keep track of the largest cluster in the system. If this cluster consists of at least 100 particles, it is a stable crystal, and the simulation can is considered crystallized. This 100 particle limit is arbitrary, but should be large enough to only allow stable nuclei to be counted, because of the strict detection limits. As an example, Radu and Schilling[24] use a cluster size of 80 as limit, without stating other parameters, but measure at high densities (where the critical nucleus size is smaller).

The program to analyse the data was originally written by Hermes[14], and has been adapted for this thesis to read the output files as produced by this authors program. No further modifications were needed, and output of the Q6-parameter proved to detect crystals efficiently.

3.3 OpenCL and computational methods

When attempting to port a simulation to the graphics card, one will quickly find that there are several bottlenecks that are either not present or less obvious on a common CPU. For an MD simulation, the main limitation is the transportation of data between the CPU and GPU, which depends on the host system (in most cases limited by the PCI-E bus). This transport may be fast, but the bandwidth is insufficient even for small simulations. The solution here is to move all data to the GPU once, and rely on the memory of the graphics card. Only after the simulation is done one can move data back.

Another problem is the processing power of a single GPU core. This limit becomes visible when a bit of code can only be executed as a single thread, for example arranging a certain memory space. In such cases the programmer has to decide between pumping all data back to the CPU which can do the task fairly quickly, or writing a kernel for it that can do the task on a GPU, leaving all but one of its cores idle. This of course is an unwanted situation, and it has been a focus of this thesis to write the program in such a way that parallel execution of almost all code is easily achieved. Basic Molecular Dynamics simulations by nature lend themselves well for massive parallel execution (as opposed to Monte Carlo simulations for example, although those too have been rewritten for graphics cards[25]), but additional features are less straightforward to execute in parallel.

In this thesis, one of the bottlenecks has been creating and updating the cell list, which requires memory pointers to be set after another, and only limited parallelism has been achieved here. However since this step is not computationally heavy, it only has a limited impact on computation times. Most time is actually spent calculating the particle-solvent interactions, and in the SRD step. These steps are fortunately embarrassingly parallel, thus maximal efficiency is easily achieved. However, graphics cards tend to run several hundreds of threads simultaneously (the used HD7950 card uses 1796 threads), thus one should make sure sufficient work can be done together, which may not be the case for methods targeting only the nanoparticles. The efficiency curve of those methods only flattens after about 10.000 nanoparticles. The amount of liquid particles is a factor 100 larger, and will not present any such bottlenecks.

3.3.1 Memory

Another major difference between CPUs and graphics cards is the memory access, because of the wider memory bus, and higher latencies on a GPU. Requesting 1 'word' (32 bits) of data will fully fill the memory bus, producing 396 bits of data for the GPU used in this thesis, compared to only 64 bits on a CPU. If only the one requested 'word' of this is used, the other bits are discarded on arrival, thus over 90% of the data is wasted. To use this bandwidth efficiently (a graphics card can have 15x more bandwidth than a common CPU) all values stored in memory should be aligned in the order they are used, allowing less requests. Since the latency to the main memory is so high (up to several hundred GPU cycles, compared to 20-40 for a CPU), it is also advisable to re-use data that are already in the local caches. Knowing this, one can try to minimize random memory access, and launch kernels in the order that they use memory; for example multiplying a set of vectors is best done sequentially, where both vectors are continuously stored in order as well.

Unfortunately not all memory access can be aligned, since working with a cell list will either require a full particle list resorting, or dereferencing of pointers, which both require random access, and in the case of resorting this step will have require double the memory size, as well as random writes.

Another issue is raised if 2 kernels want to access the same memory space at the same time, a so-called race condition. Only 1 kernel can have access, the other is usually left with either the wrong value, or no value at all, depending on the hardware. To prevent this from happening, the programmer either has to ensure every kernel has its own memory space and no overlap is possible, or has to use atomic operations. These operations are executed without interruptions, but they are slower due to the memory locks that have to be set. They guarantee that only 1 kernel reads and writes at a certain memory position, and the others wait in line for their turn.
A related method is the mutex, where a separate memory position is used to keep track of memory access. These mutexes can range from very simple 1-bit lock to complex ones that have a waiting queue.
On current graphics cards only integer atomic operations are possible, float atomics are not provided in the OpenCL 1.4 standard. They are available in the 2.0 standard which has yet to be implemented in hardware.

3.3.2 Cell list

To work around the problems outlined above, we have created a multithreaded cell list solution that splits the creation of the list in several discrete steps, which are all parallelised. This solution uses the particle list to keep track of the location and momentum of each particle and a cell list which indicates what particles are in a cell. A separate count-list is used to keep track of where each cell begins and how many particles it contains. The cell list itself is continuous, so where one cell ends, the next cell starts with the markers stored in the count-list. This count-list also contains the average speed, temperature and rotation vector of the cell, which is used for the SRD step. Note that the cell list is only used for the solvent particles, for the nanoparticles a different system is used, because there are far fewer of these, thus memory constraints play no role here.

The procedure to update the cell list is divided in 4 phases, which are followed by the SRD rotation procedure, because both use the same data.
- In the first phase the data from the previous iteration is erased and the random orientation vectors for the cell SRD procedure are calculated.
- In phase 2 we use a fixed number of threads (256, the maximum size of a workgroup) to loop over all particles and count how many there are in each cell. Since the graphics card supports atomic_add within a workgroup, count is kept in 1 count-list only.
- Phase 3 calculates the starting point of each cell, which is entered into the count-list. This happens in 2 steps, in the first all cells assigned to a thread are summed, and this value is shared between threads. Each thread then adds all totals up to its own, so it knows the starting point in the cell list for the first of its allocated cells. It can then set the rest of the cell boundaries without further inter-thread communication.
- Phase 4 calculates the exact cell of the particle again, and in another atomic-add it reads the current high-water-mark of the cell and adds 1. It can then write the particle number to the right place in the cell list (cell offset + high-water-mark). A further improvement could be caching the cell list number somewhere in the particle list to avoid recalculating, but this is a relatively light computation.

After these phases, the SRD rotation step is performed, because this requires an up-to-date cell list.

Advantages of this method are the low memory use and flexibility in particle addressing, with disadvantages being the slower access (the dereferencing of pointers takes time, and the random memory access is unfavourable), and the more complicated update routine.

The nanoparticles are treated in a different way. Here a Verlet list used, because of efficiency reasons.
In a GPU interactions are computed separately, the interaction from nanoparticle to solvent is calculated at a different time than the interaction from solvent to nanoparticle, due to memory access constraints.
The nanoparticle routine loops over all nearby solvent cells, after which the solvent cell adds the nanoparticle to its Verlet list to do the other half of the computation in the next step.
This eliminates the location mapping of all nanoparticles, or worse: an exhaustive search for nanoparticles within range. It is still not an optimal solution, but since atomic float-operations are not permitted (yet) on graphics cards, one cannot directly write the result to both locations without risking race conditions.

3.3.3 Random number generator

A graphics card unfortunately has no built-in random number generator (RNG), nor an easy routine to provide random numbers. In this thesis we generate pseudo-random numbers in a sequence using:

$$R_{next} = |\ \mathrm{modf}(a \cos(b\ R_{last}))\ | \qquad \text{with } a = 100,\ b = 200\ . \tag{3.4}$$

Modf is a method to split a number in an integer and a fractional part, and here only keeps the fractional part of the input. However, it retains the sign of the original number[18]. This RNG is slow, but a quick spectral test shows no correlations between sequentially generated numbers, and somewhat equidistributed numbers (if the constants a and b are too low, higher numbers have a slightly higher chance of occurring). This implementation however does not generate a deterministic series, due to rounding errors and precision limitations. Secondly, and more crucially, it does not pass the SmallCrush[26] test (of TestU01), which means that it cannot be seen as a reliable RNG. We decided to still use this instead of a better random number generator because it does not require any persistent memory for its internal state (it has none), and because of its computational simplicity. For our use its randomness is sufficient, it is implemented to avoid repeating the same pattern in the SRD rotations, but the method does not rely on the random numbers as much as for example Monte Carlo simulations.
To increase reliability, a better implementation would be a small-state RNG. However, linear congruential generators (which only have 1 number as internal state) are vulnerable to formation of planes[27], which means that in multiple dimensions (sometimes as few as 2 or 3) the numbers are correlated. The Mersenne Twister[28], which is the current scientific standard, has a very large internal state of 640 floating point numbers, and is thus less suited to GPUs because of the limited memory available per kernel. A WELL512[29] RNG would be a good compromise, but has not been implemented due to lack of time.
In this thesis the initial random number per iteration is generated by the CPU, and then either used directly or passed as separate 'random' number as seed to the GPU, avoiding build-up of errors.

3.3.4 Precision

An important factor in speed is the precision with which calculations are made, this can either be float (32 bits) or double (64 bits). Half-precision (16 bits) is generally not used in scientific calculations, because this will only give an accuracy of 5 decimal digits per calculation. Since errors multiply, a longer simulation is more likely to run into errors (if unchecked) than a short one, see for example the calculations done by Anderson et al[20] showing that double precision velocity Verlet schemes deviate from expected value after twice the time it takes for single precision calculations to develop a significant error. We thus in this thesis use double precision.
A small difference was found between a GPU and CPU implementation, because GPUs may not always use IEEE754 standards, but will use the indicated precision of 64 bits (CPUs use 25% extra bits while calculations are not exported to the main memory[30][31]).

The used Verlet scheme does not have energy runaway in the long term, but is still sensitive to errors due to the potential that rapidly decreases with distance. Thus, double precision (which gives roughly 16 digits accuracy) is favoured, even if it comes at a higher computation cost. This cost is limited on CPUs, all of which are 64 bits wide by design today, but graphics cards still work with 32 or even 24 bits wide designs. The logic circuits to combine some of these into double precision units cost energy, design time and chip real estate, all of which come at a premium. Therefore, only high-end chips are generally equipped with a relevant number of 64 bit units, whereas other GPUs only have a few units for compatibility reasons. Thus double-precision computation power varies from card type to card type, from half the floating point power in the Nvidia GTX 480 (which is artificially capped at 1/8)[32] and Quadro 6000, 1/3 in the modern Nvidia Titan[33] and Quadro K6000, and 1/24 in most other Nvidia cards. AMD at this moment can use ¼ or 1/8[34] of its 32-bit power in double precision on its high-end cards, and 1/16 in lower end cards.

In this thesis we used an AMD Radeon 7950, with 3 GB memory, and a manufacturer-elevated clock speed of 950 MHz (normal 800 MHz). This card has a maximum double precision processing power of 800 GFLOPS.

4 RESULTS

This section deals with the analysis of our SRD solvent, as well as the nucleation results obtained in it. To ensure the solvent behaves as expected, we tested Poisseuille flow. We then simulated nanoparticles in the liquid, and measured the velocity autocorrelation function (VACF) and diffusion, which was compared to the work of Padding and Louis[7].

We then increased the nanoparticle densities, and measured the effects of the crystallisation parameters, as well as the radial density function G(r), both of which confirmed that nucleation was occurring in our simulations. The nucleation rate itself was measured over a range of densities, and compared to other works.
Lastly we measured the simulation speed increase of using a graphics card, and the power consumption differences between CPUs and GPUs.

4.1 SRD properties

In The flow velocity of a liquid in a pipe is determined by its distance from obstructing objects, such as walls. To graph the flow, an elongated box with SRD fluid was placed between 2 walls that reflect the fluid on contact, thus any particle colliding with the wall has its momentum reversed.

4.1.1 Poisseuille flow

To verify the behaviour of the SRD liquid, we examined the behaviour of an SRD liquid under Poiseuille flow. In particular, we examined the flow of an SRD liquid confined between two walls. The expected profile of the flow can be determined from the the Navier-Stokes equation for incompressible fluids, which is given by

$$\rho(\frac{\delta v}{\delta t}+v\cdot\nabla v) = -\nabla p+\mu\nabla^2 v+f \quad , \tag{4.1}$$

where ρ is the density, μ is the viscosity, p is the pressure, v is the flow velocity, and *f* includes all other external forces acting on the system, such as gravity. Solving this equation for a planar geometry, it can be shown that the flow velocity is given by

$$v(x) \propto -\frac{1}{2\mu}(Lx-x^2) \quad , \tag{4.2}$$

where *L* is the distance between the walls. Note that this assumes that the walls are perpendicular to *x*.

To test our system, the SRD liquid was put in a box of 96 by 32 by 32 a_0^3. To create flow, all particles are accelerated by $0.005a_0$ per t_0. After the system reached an equilibrium, the flow profile displayed the expected parabola, as shown in Figure 3. This indicates that the SRD liquid behaves as desired, with little compressibility, in general obeying the Navier-Stokes equations. Note however, that the result differs somewhat from the theoretical prediction, in particular our results are shifted upwards slightly. We suspect that this arises because the SRD rotations are executed around the average velocity of a cell, which thus leaves the forward motion intact. The interactions with the wall may not be strong enough to completely negate the forward velocity that is added each time step.

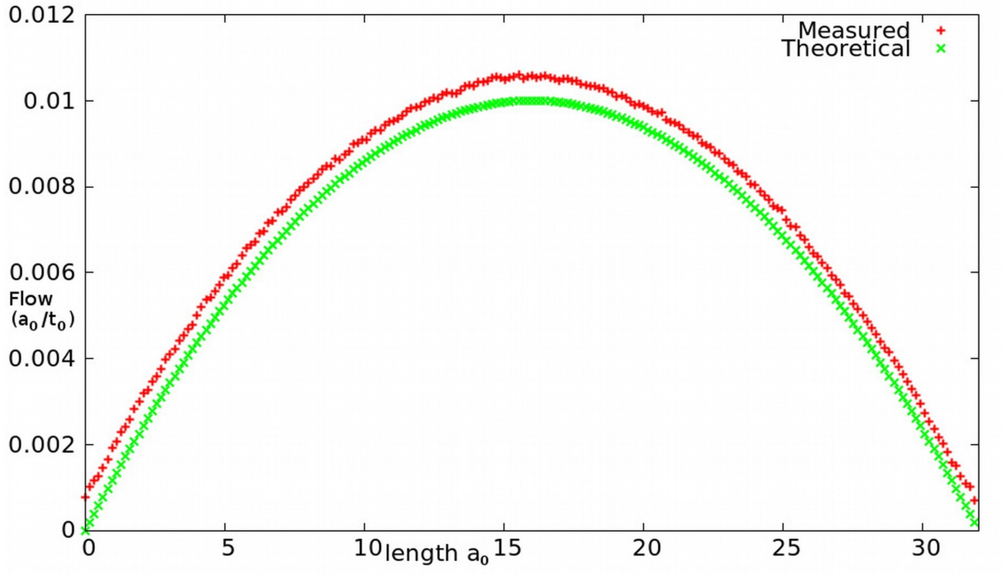

Figure 3. Poisseuille flow through 2 fixed plates spaced 32 a_0 apart. A flow of 0.005 a_0 per t_0 is applied to particles.

Despite the offset at the edges, we conclude that our SRD implementation is sound, because the profile itself is correct, with no adverse effects from incorrect randomisation, or anisotropic effects.

4.1.2 Velocity Autocorrelation Function

Once one has a proper liquid, nanoparticles placed in it should exhibit Brownian motion when no external force is applied. This means that the velocity vector should have a random direction and length after a short time, and should average to zero. To test this diffusive behaviour of nanoparticles in the SRD liquid, we measured the Velocity Autocorrelation Function (VACF). The VACF gives the correlation between the current velocity *v(t)* and the velocity at a time *t*-Δ*t* (*v*(*t*-Δ*t*)) averaged over all particles:

$$C_v(\Delta t) = \frac{1}{N}\sum_{n=1}^{N} v_n(t)\cdot v_n(t-\Delta t) = \langle v(t)\cdot v(t-\Delta t)\rangle \quad . \tag{4.3}$$

The values can vary between $v(t)^2$, which corresponds to the situation where the particle has not changed direction, and $-v(t)^2$ which corresponds to the situation where the particle has turned 180 degrees. When Δt goes to infinity, the expected value is 0 as there should be no correlation between the velocities.

In Figure 4, we plot the VACF for several different packing fractions for nanoparticles interacting with a WCA6-12 potential. All measurements were done in a box of size 64^3 a_0^3. In all cases, the nanoparticles were found in be in a liquid state as measured by our order parameter. Note that the VACF displays the expected behaviour, namely that at low densities it decays exponentially, and for all densities approaches 0 in the long time limit. At the highest two densities, corresponding to packing fractions of 0.45 and 0.5, we observed some fluctuations below zero. This is common for high density liquids and is typically attributed to backscattering.

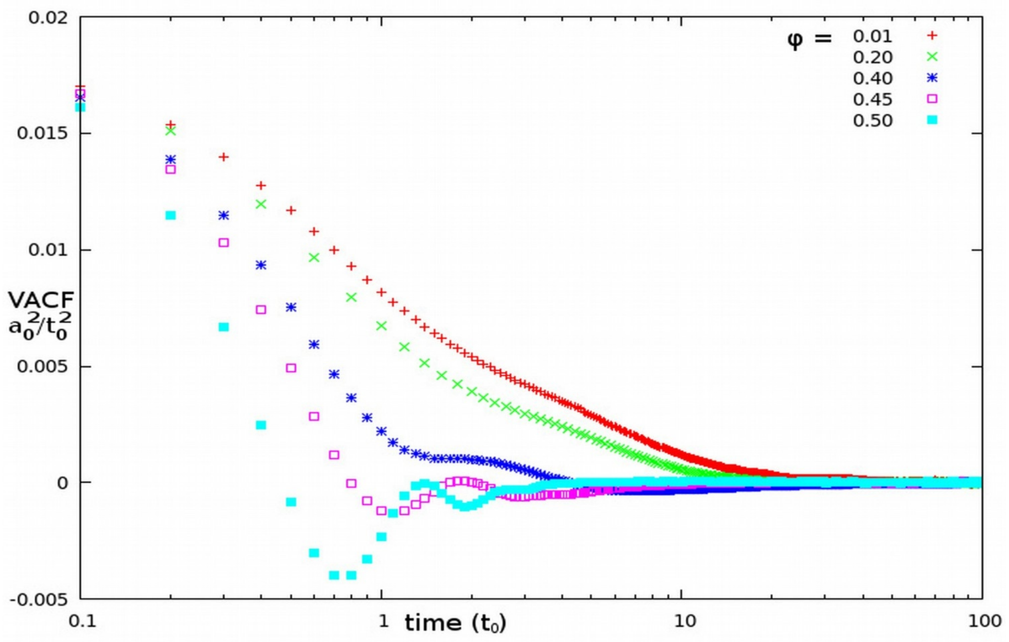

Figure 4. VACF for multiple concentrations of nanoparticles, measured in a box of 64^3 a_0^3. At low packing fractions, the interactions with the solvent dominate, while at the higher fractions particle-particle interactions cause oscillating behaviour.

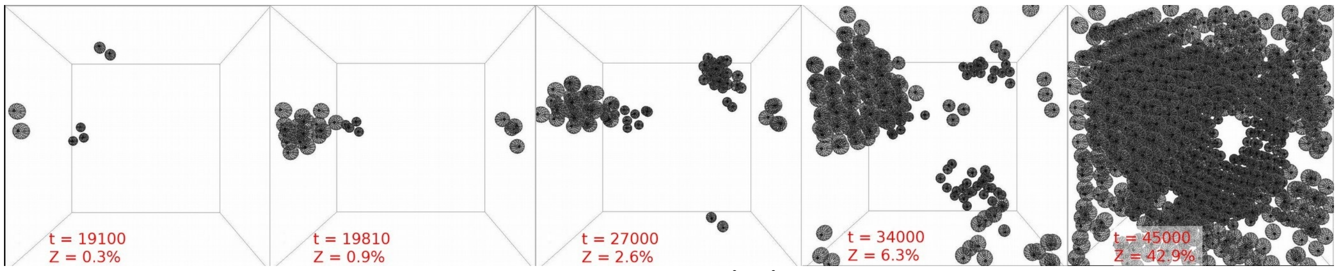

Figure 5. Nucleation example over time (t in t_0) in a box of 64^3 a_0^3, only crystalline particles are shown. Due to periodic boundary conditions the crystal expands into opposing sides of the box.

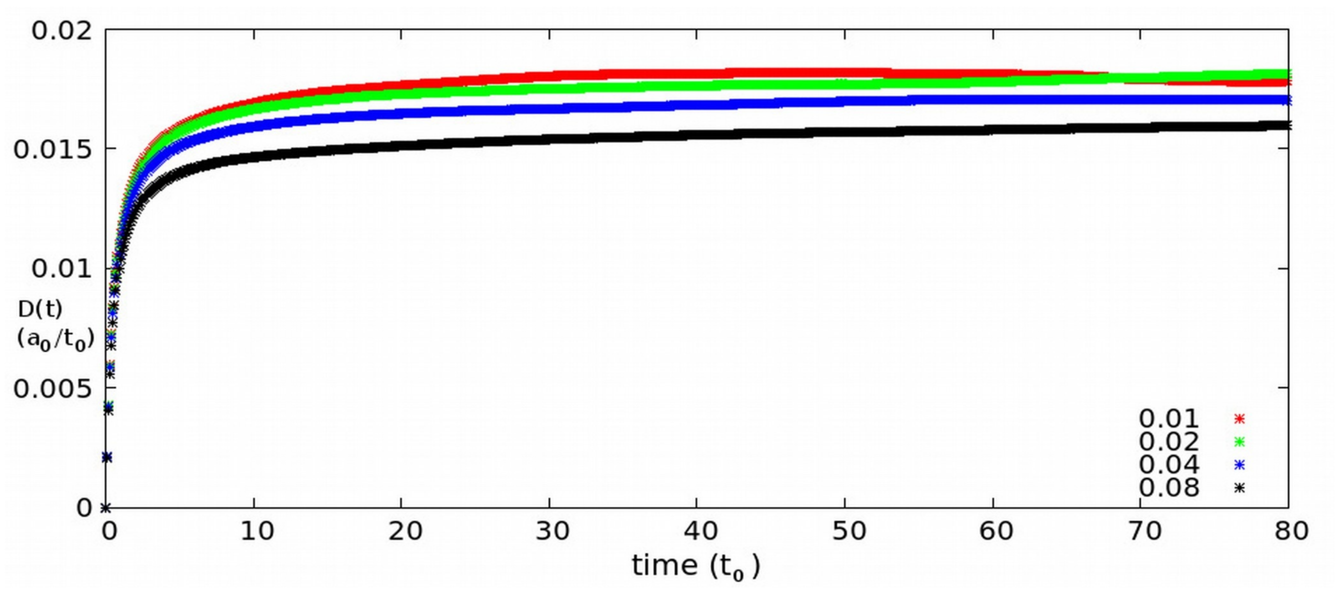

Figure 6. Time-dependent diffusion coefficient of nanoparticles in SRD for various low packing fraction. Initially diffusion patterns are ballistic (exponential increase) but it levels off until the long-term diffusion coefficient D is reached.

4.1.3 Diffusion

Two common methods can be used to determine diffusion coefficients. The first is based on the VACF and uses Green-Kubo relations. According to Green-Kubo one can derive a transportation coefficient (such as the diffusion) from the associated autocorrelation function. In the case of diffusion, this is given by

$$D = \frac{1}{3N}\int_0^\infty \langle \sum_N v(t)\cdot v(0)\, dt \rangle \quad . \tag{4.4}$$

Secondly, instead of measuring the VACF, we can relate the diffusion coefficient to the mean square displacement by

$$D = \frac{1}{2d}\lim_{\tau\to\infty} \widetilde{D}(\tau) \quad , \tag{4.5}$$

where

$$\widetilde{D}(\tau) = \frac{\langle (x(\tau)-x(0))^2 \rangle}{\tau} \quad , \tag{4.6}$$

with x*(t)* the position of the particle at time *t*, and *d* the number of dimensions considered.

Using the VACF, we measured diffusion at a packing fraction of 0.01, which is low enough to allow almost unobstructed behaviour, while there are still enough particles to obtain a reliable average. This gives us an approximation for the diffusion in the infinitely dilute limit, typically denoted D_0.

We found at this packing fraction, a diffusion coefficient of 0.022 a_0^2/t_0. This is reasonably in agreement with the results from Padding & Louis who found $D_0 = 0.02\ a_0^2/t_0$.

We also examined the diffusion by measuring the time-dependent diffusion coefficient $\tilde{D}(\tau)$. We used a large box (100 a_0 on all sides) and simulated for 2500 t_0. We examined low packing fractions of 0.01 to 0.08, where the diffusion should be close to D_0. The results are plotted in Figure 5. The measured diffusion coefficients tend to fluctuate even over long periods of time, and we had trouble getting reliable results. Runs in a box of 64^3 a_0^3 proved unsatisfactory, and it took a box size of 100^3 a_0^3 before all lines were in the expected order, e.g. higher densities should have lower diffusion coefficients. Note that the diffusion coefficient we obtain for a packing fraction of 0.01 is slightly smaller than what we predicted via the VACF.
From these results we measured a D_0 of around 0.017 a_0^2 per t_0, which is below the measurements from VACF as well, but still within acceptable margins.

4.2 Crystallisation detection

After testing the liquid and the nanoparticle behaviour, we have tested the parameters as used in the nucleation detection. Our focus was on the bond cut-off of the q_6q_6-parameter, which can be varied between 0 and 1, in which case either everything is considered a bond, or only particles in exactly identical environments share a bond. We simulated a box of 64^3 a_0^3 for 25k t_0, and recorded positions every 10 t_0. We then computed the q_6q_6 for a bond cut-off between 0.5 and 0.8, and measured the crystallinity of the system. Figure 7 shows the largest cluster measured for these cut-offs, which is used to determine whether we have a stable nucleus.

The trendline is clear, the higher the cut-off, the lower the cluster size. Also visible is that the amount of noise is reduced significantly with higher cut-off, thus we use a cut-off of 0.7 throughout the thesis. For this cut-off, we found a nucleus of 75 particles stable, and from observing our simulations we concluded that such a nucleus will in all cases continue to grow.

The fraction of crystalline particles keeps increasing after a nucleation event, however the pace is usually slow, and it can take up to 50k t_0 before the box is >80% crystalline (due to edge defects, 100% is impossible to reach). Due to this slowness 'nucleation' is considered to have happened once the maximum cluster size reaches 100.

To check whether we actually obtained a crystal, the radial distribution function, G(r) is measured. This function gives the amount of particles found plotted against the distance from the nucleus.
We used the production runs for these plots, with 3200 nanoparticles in a 64^3 a_0^3 box, in different stages of crystallisation. The results is shown in Figure 8.

The radial distribution function of the liquid shows that at these high packing fractions there is already significant ordering, although at larger interparticle distances than expected in a crystal. When an actual crystal is forming, the peaks become more pronounced. The peaks suggest an FCC structure, although long range order is not yet present.

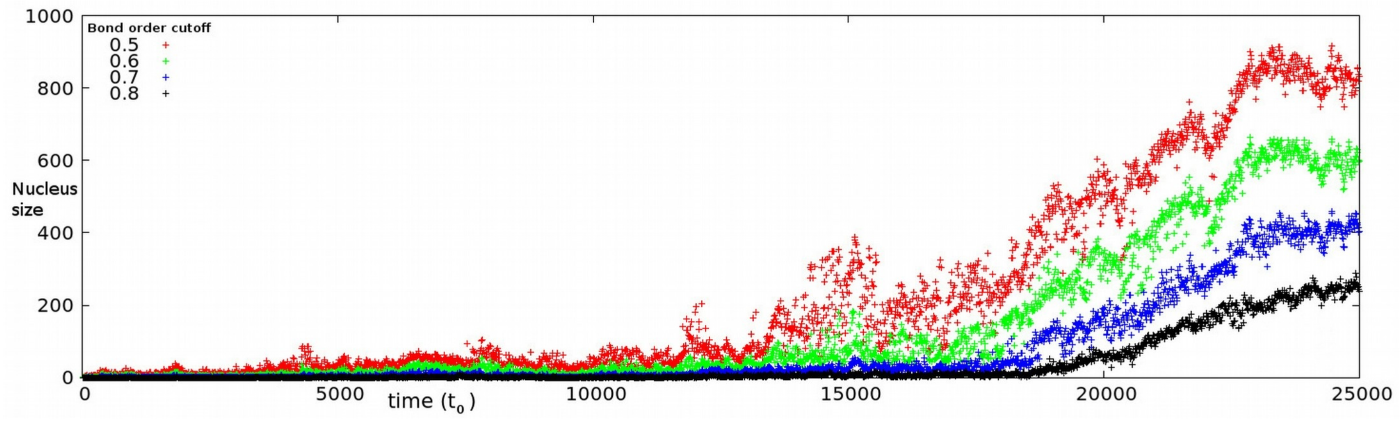

Figure 7: Largest nucleus for different bond order cut-offs as a function of time. Nbond is fixed at 8, and the nbond-cut-off at 0.7. A lower bond order cut-off means more particles are considered bonded, and thus more particles are considered crystalline. Nucleation happens around 20.000 t_0 in this figure.

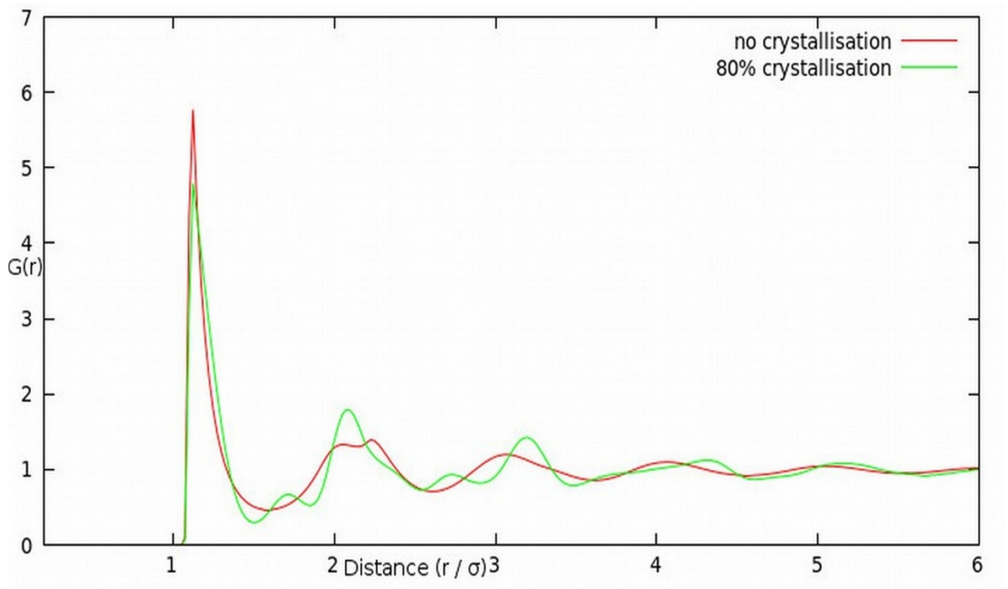

Figure 8: Density function G(r) for crystallized and non-crystallized particles, the particle density φ is 0.41 in both cases.

4.3 Crystallisation as function of packing fraction

The main goal of this thesis is to measure the nucleation rate as function of packing fraction. To do so, we used a box of 64^3 a_0^3, and ran a simulation in it for a given time, varying from 15k t_0 to 50k t_0, and afterwards scanned the results to find the largest cluster. Unfortunately due to computer errors several simulations were aborted prematurely. For all packing fractions we sum the time until either a nucleation event was observed or until the time ran out, and divide this by the number of nucleations observed. At the lowest 2 densities no nucleation occurred in the 550k t_0 we simulated. The results are shown in Table 2. When plotting the nucleation rate against packing fraction φ (see Figure 9), it becomes clear that the nucleation rate follows a near-exponential relation with packing fraction in the region 0.395 to 0.41, which is as expected. Above this the dependence mostly vanishes, and at larger packing fractions the rate actually drops a bit (this happens roughly from $\varphi = 0.425$ and above, not shown). Below 0.395 however no crystallisation has been observed, and nucleation rates become very small. Thus a total runtime of well over 10M t_0 would be required to obtain statistically significant results. Error margins in all cases are significant, around 1 order of magnitude at the lowest data points, and around 20% for higher values.

In Figure 10, we compare our nucleation rates to those in literature. Note that for this comparison, we have rescaled the rates so that the time is now measured in D_0^{-1} to correct for nanoparticle size. Comparison with other simulations (see Figure 10) shows that the SRD results mostly follow the hard sphere experiments of Sinn et al[35], and disagree with the Brownian Dynamics and Forward Flux Sampling as done by Filion & Ni[2]. In both of these works, a WCA potential with $\beta\varepsilon = 40$ and an implicit solvent was used. The main difference thus is the type of solvent used, which suggests that this may be responsible for the steepness of the nucleation rate.
The hard upper edge of 10^{-4} is because creating a metastable liquid is not possible for such supersaturated systems. Thus a system would either be nucleated from the start, or nucleate after the start-up phase has been completed.

$\rho\sigma^3$	φ	N	total length (t_0)	# nucleations	Rate ($1/t_0$)
0.745	0.390	3050	250000	0	0.0
0.751	0.393	3075	305700	0	0.0
0.757	0.396	3100	1011700	5	4.9E-6
0.763	0.399	3125	403300	5	1.2E-5
0.769	0.403	3150	146650	10	6.8E-5
0.775	0.406	3175	54400	4	7.4E-5
0.781	0.409	3200	82800	9	1.1E-4
0.787	0.412	3225	40200	4	1.0E-4
0.793	0.415	3250	35300	4	1.1E-4
0.800	0.419	3275	34400	5	1.5E-4
0.806	0.422	3300	76500	7	9.2E-5

Table 2. Overview of nucleation rate for a given number density $\rho\sigma^3$ or packing fraction φ, number of particles N, amount of nucleations observed for a total length of simulation time.

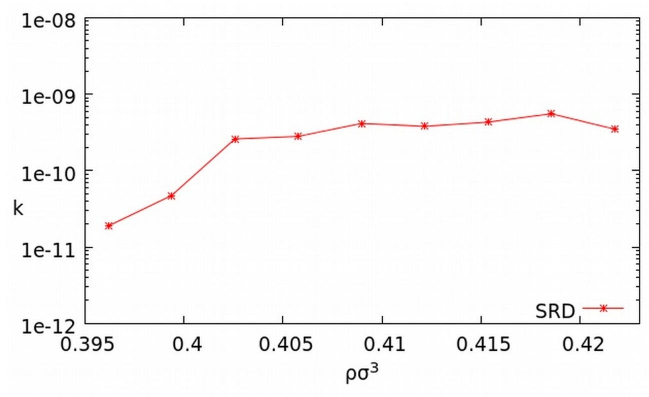

Figure 9: Computed nucleation rate k as function of packing fraction, for a box size of 64^3 a_0^3.Note that k is determined per a_0^3.

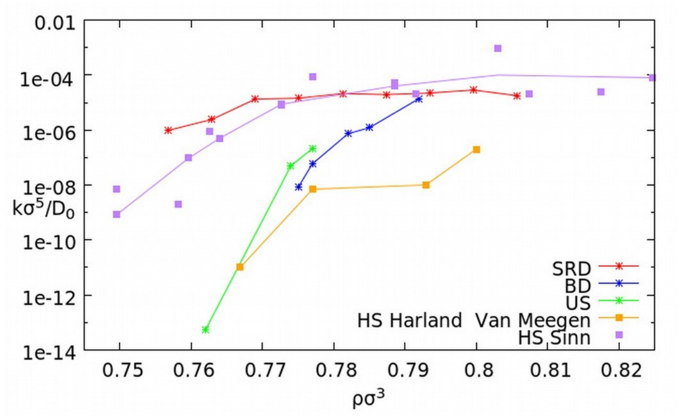

Figure 10: Reduced nucleation rate $k\sigma^5/D_0$ as function of number density $\rho\sigma^3$ measured with Stochastic Rotation Dynamics, Boltzmann Dynamics[14] and Umbrella Sampling[14]. Also included are the hard sphere experimental results from Harland & van Meegen[4] and Sinn[35].

4.4 OpenCL speed increases

Although the main focus of this thesis is on nucleation rates, a significant effort was made to optimise the program for use with a graphics card. The results will be discussed here.

Comparing efficiency between computer systems is hard, and numbers obtained from one set of hardware cannot easily be extrapolated to other sets. The efficiency of GPUs varies greatly per task given[36], and program run times vary from architecture to architecture, and from operating system to operating system. Small differences between different runs on the same machine are also observed, for example if the computer is multitasking, has had a fresh reboot, and the presence of a flexible clock speed can all affect the computation time. When comparing graphics card speed, all these factors come into play, as well as a few new ones. For example, since a graphics card runs hundreds of threads at the same time, it will be most efficient when simulating a large number of particles, which requires large systems. In this thesis a comparison is made for two system sizes, for run time per unit (1 CPU or 1 GPU) and for efficiency in iterations per kilowatt.

All tests are done using a packing fraction $\varphi = 0.41$. The box is primed and equilibrated as in a normal nucleation run, and mid-run we measure the time taken to complete 100 t_0 (1000 SRD-iterations). Power draw is estimated based on manufacturer thermal design power specifications, which are the maximum expected loads, and most likely overestimate the actual power draw[37]. The tests were ran on a system containing both an AMD Phenom 1090T CPU and the AMD Radeon 7950 GPU, and on the university cluster. This cluster has many different types of processors, as well as a varying load due to other users, so an average will be taken here for each of the 2 main types of CPU, which are named 2350 (AMD Opteron 2350) and 6238 (AMD Opteron 6238).

Speed test	$\varphi = 0.41$				
Time required for 100 t_0			Box size 64^3	Box size 128^3	Efficiency
Type	Name	Power draw	Time (seconds)	Time (seconds)	(t_0/kW)
GPU naïve	7950	160W / card	414	2535	15.10
GPU Optimized	7950	160W / card	214	1563	29.20
GPU Float	7950	160W / card	140	1249	44.60
CPU	1090T	125W / 6 cores	9920	118660	4.83
CPU	1090T (no cell list)	125W / 6 cores	1158000	not measured	0.04
CPU	Mars 2350	95W / 4 cores	11000	252590	3.83
CPU	Mars 6238	115W / 12 cores	16000	301460	6.52

Table 3: Computation speed for different box sizes, and power efficiency for each of the configurations used. 7950 is the AMD Radeon 7950 graphics card, 1090T is an AMD Phenom II CPU at 3.2 GHz, and the Mars systems are an AMD Opteron 2350 and a 6238 respectively. Times are given per core.

Optimisations were used where possible, which for the CPUs meant using the O3 set from the gcc compiler. OpenCL as of now lacks such optimisations, and here we tested a naïve implementation (where data is passed as simple structures), as well as an optimised version. This version used vectorisation of the data (so it can be stored and processed as a package of connected values), and reduction of data transport between the GPU and the CPU host. This increased GPU efficiency by almost 50%.

Table 2 shows that even a basic graphics card simulation for the 64^3 a_0^3 box is roughly 25 times faster than the CPU cores tested, although said CPUs have multiple cores. Comparing the 1090T with a 7950 with optimized code shows a clear advantage for the graphics card in terms of absolute speed (46x faster compared to a single CPU core). When comparing efficiency (in iterations/watt), the graphics card is still leading, although with a smaller margin. Other CPUs tested have similar power draw per Floating Point Operations Per Second (FLOPS), though divided over multiple cores.

As another test, we ran the CPU code without a cell list. From these results, it is clear that a cell list or other way of reducing interaction computations is an absolute requirement for a faster simulation, with a speed advantage of 100x obtained by its implementation.

Increasing the box size to 128^3 a_0^3 and 25600 nanoparticles, which keeps packing fraction at 41%, the graphics card becomes 50 to 200 times faster than a single CPU core, mostly because its parallelism now reaches full efficiency (3200 nanoparticles is not enough to keep all 1796 cores of the graphics card continuously busy). This is in line with expectations: for simple code a speedup of 50 to 100 times can be expected, whereas more complex code, with inter-thread dependencies will profit less from running on graphics cards.

Running all simulations with floating point precision (32 bits) instead of double precision (64 bits) was tested as well. On CPUs the difference between the two is negligible, but graphics cards do show a significant difference, especially for smaller size boxes. Graphics cards have a much higher 32-bit FLOPS than 64-bit speed by design (see Section 3.3.4). Note that the difference remains well below the theoretical maximum of 4. This may be due to the difference in data transportation speed compared to the computation speed.

5 CONCLUSIONS

In this thesis we studied the nucleation of WCA particles using a coarse-grained explicit solvent model, SRD. Before examining the nucleation, we examined the properties of both the SRD fluid and the nanoparticles dissolved in the SRD fluid. For the SRD fluid itself we found it behaved as a proper Navier Stokes liquid. We also measured the velocity autocorrelation and the diffusion coefficient in the dilute regime, which were used to approximate D_0. Both methods gave a diffusion of approximately $D \approx 0.02$, which agrees fairly well with that of Padding and Louis[7] who studied a similar system.

As our main research subject, we measured the nucleation rate for nanoparticles. We found a significant difference to the nucleation rate predicted by simulations without an explicit solvent. In particular, the lower packing fractions deviate by many orders of magnitude, which cannot solely be attributed to inaccuracies in measurement. This difference is most likely explained by the type of solvent used. Comparing to experiments, we find better agreement with the nucleation rate of hard colloidal particles. However here we find that we cannot completely sample the lower packing fractions due to a lack of computational power.

Improvements can be made in the simulation, perhaps adding biased sampling methods (such as described in Sec 2.2.2) to investigate even lower packing fractions, and improvements on the random number generator. The latter would increase robustness of the SRD fluid. We found the fluid to produce the correct behaviour in our tests, but small deviations in our RNG may produce undesired side-effects in long term simulations.

The results obtained in this research suggest that the crystallisation as function of packing fraction is dependent on the method used to simulate the solvent, and in this case, an explicit solvent seems the best option available. Fully simulating every solvent particle is still too computationally demanding, thus a compromise can be reached in coarse-grained solvents such as SRD.

Running simulations on a graphics card has proven to be a successful way to speed up the computation of embarrassingly parallel programs, at the expense of additional programming time. The improved power per watt is an additional advantage, but the drawback is that new hardware must be purchased. The changes required in programming style should not be prohibitive to a wider implementation of graphics cards for high-performance computing.

BIBLIOGRAPHY

[1] Alder B.J., Wainwright T.E. ,*Phase transition for a hard sphere system*, J. Chem. Phys. 27 (1957) 1208-1209
[2] Filion L., Ni R., Frenkel D., Dijkstra M.,*Simulation of nucleation in almost hard-sphere colloids: the discrepancy between experiment and simulation persists*, J Chem Phys 134 (2011) 134901
[3] Kawasaki T., Tanaka H.,*Formation of a crystal nucleus from liquid*, PNAS 107 (2010) 14036-14041
[4] Harland J.L., van Megen W.,*Crystallisation kinetics of suspensions of hard colloidal spheres*, Phys Rev E. 55 (1997) 3054
[5] Malevanets A., Kapral R.,*Mesoscopic model for solvent dynamics*, J Chem Phys 110 (1999) 8605
[6] Weeks J.D., Chandler D., Andersen H.C.,*Role of Repulsive Forces in Determining the Equilibrium Structure of Simple Liquids*, J Chem Phys 54 (1971) 5237
[7] Padding J.T., Louis A.A.,*Hydrodynamic interactions and Brownian forces in colloidal suspensions: Coarse-graining over time and length scales*, Phys Rev E 74 (2006) 031402
[8] Ermak D.L., McCammon J.A.,*Brownian Dynamics with Hydrodynamic Interactions*, J Chem Phys 69 (1978) 1352-1360
[9] Geyer T., Winter U.,*An O(n2) approximation for hydrodynamic interactions in Brownian dynamics*, J Chem Phys 130 (2008)
[10] Horbach J., Succci S.,*Lattice Boltzmann versus Molecular Dynamics Simulations of Nanoscale Hydrodynamic Flows*, Phys Rev Lett 96 (2006) 224503
[11] Asakura S., Oosawa F.,*Interaction between Particles Suspended in Solutions of Macromolecules*, J Poly Science 33 (1958) 183-192
[12] Mullin J.W.,*Crystallisation*, (2001)
[13] Auer S., Frenkel D.,*Quantitative prediction of crystal-nucleation rates for spherical colloids: a computational approach*, Annu. Rev. Phys. Chem. 55 (2004) 333 - 361
[14] Filion L., Hermes M., Ni R., Dijkstra M.,*Crystal nucleation of hard spheres using molecular dynamics, umbrella sampling and forward flux sampling*, J Chem Phys 133 (2010) 244115
[15] Allen R.J., Valeriani C. ten Wolde P.R.,*Forward flux sampling for rare event simulations*, J Phys Condens. Matter 21 (2009) 463102
[16] Fan Z., Qiu F., Kaufman A., Yoakum-Stover S.,*GPU Cluster for High Performance Computing*, ACM/IEEE Supercomputing Conference (2004)
[17] Buck I., Foley T., Horn D., Sugarman J., Fatahlian K., Houston M., Hanrahan P.,*Brook for GPUs: Stream Computing on Graphics Hardware*, ACM Transactions on Graphics 23 (2004) 777-786
[18] ,*OpenCL online manual*, (visited 20 sept 2013)
[19] Van Meel J.A., Arnold A., Frenkel D., Portegies Zwart S.F., Belleman R.G.,*Harvesting graphics power for MD Simulations*, Mol Sim 34 (2008) 259-266
[20] Anderson J.A., Lorentz C., Travesset A.,*General purpose molecular dynamics simulations fully implemented on graphics processing units*, J Comp Physics 227 (2008) 5342-5359
[21] Wang J., Shan J.,*Space-Filling Curve Based Point Clouds Index*, Geocomputation (2005)
[22] ten Wolde P.R., Ruiz-Montero M.J., Frenkel D.,*Simulation of homogeneous crystal nucleation close to coexistence*, Faraday Discuss. 104 (1996) 93
[23] Lechner W., Dellago C.,*Accurate determination of crystal structures based on averaged local bond order parameters*, J. Chem. Phys 129 (2008) 114707
[24] Radu M., Schilling T.,*Solvent hydrodynamics affect crystal nucleation in suspensions of colloidal hard-spheres*, ()
[25] Preis T., Virnau P., Paul W., Schneider J.J.,*GPU accelerated Monte Carlo simulation of the 3D and 3D Ising model*, J Comp Phys 228 (2009) 4468-4477
[26] L'Ecuyer P., Simard R.,*TestU01: A C Library for Empirical Testing of Random Number Generators*, ACM Trans on Math Software 33 (2007) 4

[27] L'Ecuyer P.,*Uniform Random Number Generation*, ()

[28] Matsumoto M., Nishimura T.,*Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator*, ACM Trans on Modelling and Comp Sim (1998)

[29] Panneton F., L'Ecuyer P., Matsumoto M., ,*Improved Long-Period Generators Based on Linear Recurrences Modulo 2*, ACM Trans on Math Software 32 (2006) 1-16

[30] ,*754-2008 - IEEE Standard for Floating-Point Arithmetic* , (2008)

[31] Kahan W.,*Lecture Notes on the Status of IEEE Standard 754 for Binary Floating-Point Arithmetic*, (1997)

[32] Smith R.,*Nvidia's Geforce GTX 480 and GTX 470*, (mar 26, 2010)

[33] Smith R.,*Nvidia's Geforce GTX Titan, Part 1*, (feb 19, 2013)

[34] Smith R.,*The AMD Radeon R9 290 review*, (nov 5, 2013)

[35] Sinn C., Heymann A., Stipp A., Palberg T.,*Solidification kinetics of hard-sphere colloidal suspensions*, Progr Colloid Polym Sci 118 (2001) 266- 275

[36] Friedrichs M.F., Eastman P., Vajdyanathan V., Houston M., Legrand S., Beberg A.L., Ensign D.L., Bruns C.M., Pande, V.S.,*Accelerating Molecular Dynamic Simulation on Graphics Processing Units*, J Comput Chem 30 (2009) 864-872

[37] Collange S., Defour D., Tisserand A.,*Power Consumption of GPUs from a Software Perspective*, 9th International Conference on Computational Science (2009) 914-1923