Enhanced Window Sampling Using CUDA Enabled Devices

Jaffar Hasnain

January 29, 2010

Acknowledgments

I would like to dedicate this Master's Thesis to my family, whose support and love means more to me than words can express.

I am also grateful to Jürgen Köfinger, Michael Grünwald, Wolfgang Lechner, Philip Geiger, and Georg Menzl for being colleagues that I admire and friends.

Finally I would like to thank Professor Christoph Dellago for creating a work environment in which scientific discovery is undertaken with a sense earnestly that does not compromise with the enjoyment of the subject; and for letting me be a part of it.

Abstract

This Master's thesis is essentially a proof of principle that aims to address the question as to whether a CUDA enabled GPGPU (General Purpose Graphics Processing Unit) is capable of conducting simulation experiments that are relevant to computational physicists. Two projects found on the NVIDIA website [1,2] and a publication by J.A. van Meel [3] serve as the inspiration for this undertaking.

The system that is examined is the starting point of the detailed analysis of the nucleation mechanisms of undercooled Lennard-Jones liquids as they transition crystalline phase. The predictions of classical nucleation theory shall be validated by measuring the free energy as a function of the size of of the crystal nuclei that form in the undercooled liquid. The work done by Lechner [4] on the nucleation of the Gaussian core model serves as the touchstone of this thesis while the original architects of the analysis are Frenkel *et. al.* [5] and Daniele Moroni [6]. Accordingly, bond order analysis and umbrella sampling algorithms had to be developed in CUDA in addition to a Metropolis Monte Carlo simulation. The complexity of the algorithms required for the investigation of the nucleation of the Lennard-Jones liquid are considered an appropriate benchmark for the performance of GPGPUs in modern physics simulations.

It was found that a Tesla C870 GPGPU, despite the limitation of only being able to use floating point operations, was able to accurately reproduce the results obtained by Moroni and Frenkel *et. al.* and that the computational time was reduced by at least an order of magnitude compared to a Intel$^{\mathrm{R}}$ Core$^{\mathrm{TM}}$ Duo 6600 , making the GPGPU as efficient as a cluster of such CPUs.

Contents

1 Introduction **1**

2 The Nucleation of Lennard-Jones Liquids **3**
2.1 Classical Nucleation Theory and Rare Events 3
2.2 Bond Order Parameters and Crystallinity 6
2.3 Umbrella Sampling 9
2.3.1 Introduction to the Umbrella Sampling Technique 9
2.3.2 Bias Switching 12
2.4 Summary 13

3 The Hybrid Monte Carlo Algorithm **15**
3.1 Brief Overview of the RDMC Algorithm 15
3.2 Motivation for a New Monte Carlo Move 19
3.3 A New Trial Move 19
3.4 Efficiency Considerations 20
3.4.1 Optimizing trajectory lengths 21
3.5 Sampling Other Ensembles 22
3.6 Summary 24

4 Implementation of CUDA Functions **25**
4.1 Force-Type Functions 27
4.1.1 Implementation of the Force Function 28
4.1.2 CUDA Specific Programming 32
4.1.3 The FillNeighbourLists Function 36
4.2 Embarrassingly Parallel Functions 37
4.2.1 The Integration Function 37
4.2.2 Other Embarrassingly Parallel Functions 39
4.3 Cell List Algorithms in CUDA 39

5 Results **41**
5.1 Hybrid Monte Carlo Simulations 41
5.2 Nucleation Results 45
5.2.1 NPT-Simulations 45
5.2.2 Bond Order Parameters 47
5.3 The Free Energy of Nucleation 49

5.4 Speedup Factors 51
5.5 Summary 56

6 Final Remarks and Outlook **57**

A Appendix **61**
A.1 Zusammenfassung 61

List of Figures

2.1 The free energy curve of the formation of crystal nuclei in an undercooled Lennard-Jones liquids as calculated in Reference [6]. 5
2.2 The pair correlation functions for a Lennard-Jones liquid, fcc crystal, and bcc crystal close to coexistence [6]. 6
2.3 Distribution of q_6 order parameter for liquid, fcc and bcc phase close to coexistence [5]. 8
2.4 Distributions of the $\vec{q}_6$ dot products and the number of connected particles 9
2.5 A typical configuration that was generated by the simulation. 10
2.6 Unbiasing and pasting the free energy curve 12

3.1 A two-dimensional depiction of an RDMC move. 19
3.2 Autocorrelation time of the potential energy of a Lennard-Jones system vs the number of MD steps employed per HMC step [7] 21
3.3 Autocorrelation time of the potential energy of a Lennard-Jones system vs the discretization size δt 23
3.4 HMC acceptance rates as a function of the discretization step, δt, for $N_{MD} = 10$. The shortest autocorrelation time of the potential energy is obtained for $\delta t = 0.1$ corresponding to an acceptance rate of 70%. 23

4.1 CUDA Visual Profiler summary of the functions employed in the simulation versus the percentage of the total time that was required by each function. The values that are in parenthesis next to the function names are the number of times the function was called. 27
4.2 The symbolic decomposition of each NPT ensemble into a series of cells that are processed concurrently by the GPGPU's various multiprocessors (each of which is designated as an MP in the figure) during the calculation of the Lennard-Jones forces. 28
4.3a Basic structure of the force function. 30
4.3b Continuation of the basic structure of the force function. 31
4.4 A force function for which memory-write conflicts occur. 33
4.5 Correct implementation of the force function. 34
4.6 The integration function presented schematically. 38

5.1 Comparison of HMC acceptance rates between our and Mehlig *et. al.*'s implementations for various discretizations of the equations of motion δt. The "comparison curve" (dotted line) was extracted from Reference [7] and the "simulation results" were obtained from the author's program. 42
5.2 HMC acceptance probabilities 43
5.3 Density fluctuations as a function of time 46
5.4 q_6 distributions 48
5.5 Distribution of the dot products between nearest neighbors 48
5.6 Comparing the distributions of solid bonds for $r_c = 1.4\sigma$ 48
5.7 Comparing the distributions of solid bonds for $r_c = 1.5\sigma$ 49
5.8 The free energy of nucleation of a Lennard-Jones liquid at $T = 0.89$ and $P = 5.68$. The comparison curve has been obtained from [6]. . . 50
5.9 Corrected free energy Curve 52
5.10 Shifted free energy curve 52
5.11 Runtime vs number of simulation windows 53
5.12 Calculation time for 10000 nucleation iterations as a function of the number of basecells for multiple system sizes. 55
5.13 Speedup factor as a function of system size. 55

Chapter 1

Introduction

Since the introduction of Graphics Processing Units (GPUs) by IBM in 1981, the hardware designed specifically to render computer graphics has undergone numerous revolutions over the last two decades. Unlike CPU chips, which until recently, tend to be optimized towards evaluating a sequence of operations as quickly as possible, GPUs have steadily their capacity to execute operations in parallel. The efficacy with which GPUs complete computational tasks along with the renewed interest of the scientific and private sector in conducting computations in parallel (traditionally through the use of CPU clusters) has inspired the three largest manufacturers NVIDIA, ATI, and Intel to develop a line of General Purpose Graphics Processing Units (GPGPUs) with which all manner of numerics can be conducted.

In addition to reading about the successes of GPGPU applications on the websites of their manufacturers [8], a number of groups such as Vijay Pandai's have been able to obtain terra FLOPS of computational power with the assistance of GPUs [9]. Furthermore, as of now, most commercially available GPUs are as simple to program as GPGPUs which might mean that these devices (which are essential for any workstation) could become an integral part of numerical applications in the near future.

In order to reduce the learning curve that is required to program GPGPUs, an Application Programming Interface (API) must be designed that is compatible with the typical programming languages employed by numericists. The API that shall be examined in this thesis is the Complete Unified Device Architecture (CUDA) by NVIDIA which is an extension of the C programming language. The CUDA compiler, NVCC (which uses the the syntax of the gcc compiler) the CUDA drivers, and the CUDA programming guide can be downloaded for free from the CUDA website [10].

The structure of the thesis follows the traditional progression from general to specific, corresponding to the discussion of the physical and then gradually proceeding to the algorithmic. In Chapter 2, the general phenomenon of nucleation is defined, discussed, and the simulation techniques for its analysis are introduced. Upon determining that a Monte Carlo simulation would be appropriate, the Hybrid Monte Carlo scheme is presented as an alternative to the conventional condensed

matter Monte Carlo algorithm in Chapter 3. In Chapter 4, the programming techniques and considerations specific to the CUDA dialect are presented and illustrated. The thesis concludes with Chapters 5 and 6, the former documenting the results of the simulations that were conducted and the latter interpreting them in light of the query has been posed.

Chapter 2

The Nucleation of Lennard-Jones Liquids

Phase transitions are amongst the most dramatic and intriguing occurrences in nature. Some of the most advanced theoretical mathematics has been applied to this field of study, such as the Onsager solution [11], while careful and precise experimentation has uncovered remarkable behavior in binary systems, such as the critical Casimir effect [12]. In this numerical analysis, the nucleation of the solid phase in an undercooled Lennard-Jones liquid shall be examined. This system has been studied at great length and its nucleation is a well understood phenomenon. One of the objectives of this thesis is to reproduce the results of some noted authors [5,6].

In this chapter the physics of nucleation shall be briefly introduced and reviewed. Furthermore, the tools with which the examinations were conducted will be developed. In order to illustrate and support the arguments put forth in this chapter, extensive use of the graphs from References [5,6] is made. Since it is intended to reproduce these graphs, the specification of simulation parameters and quantitative results is postponed until Chapter 5.

2.1 Classical Nucleation Theory and Rare Events

The phase diagram of a thermodynamic system is a visualization of its equation of state. Through the demarcation of the coexistence curves one is able to trace the regions in which the different phases of the system are thermodynamically the most stable. Once the phase diagram has been mapped out, the next natural undertaking is to examine the transitions from one phase to another. The procedure is simple for such an analysis: by preparing a system in a given phase and changing the volume, temperature, or pressure one arrives at a region in the phase diagram in which the original phase is no longer free energetically favorable and a phase transition takes place.

During the course of a phase transition, the system in question forms a mixture of phases and therefore the thermodynamic analysis of multi-phase systems applies. The discussion and equations that follow have been adapted from [13]. When con-

sidering the behavior of a multiphase system that is in equilibrium, a separation of two phases occurs if the contribution of the interface between the two phases to the Helmholtz free energy is positive. The alternative to a positive contribution would, entropically speaking, drive the system towards maximizing the size of the interface, resulting in a homogenous mixture of the two phases. Furthermore, the geometry of the interface is determined by a minimization principle. If one of the phases is more dense than the other, then a two phase mixture in a gravitational field forms an approximately planar interface. In the absence of gravity, a spherical core appears since this arrangement minimizes the phase interface.

The concepts used to describe the coexistence of phases in equilibrium were mentioned in order to introduce the notion that a phase interface can have a free energy penalty associated with it. When the system changes phase, the interface penalty plays an important role in determining the characteristics of the transition mechanism. The classification of a phase transition as a nucleation event stems from the observation that it is initiated by a highly localized core of a given phase that grows to encompass the entire system. These initial nuclei are composed of particles that are in the thermodynamically stable phase but they also feature the interfaces that have been discussed. The fact that the new phase originates from a nucleus instead of appearing homogeneously throughout the system suggests that the interface between the two phases has a positive contribution to the free energy of the system. The Gibbs free energy for the formation of spherical nuclei is defined in the following manner:

$$\Delta G = G_2 - G_1 = 4\pi R^2 \gamma + \frac{4}{3}\pi R^3 \rho_1 \Delta\mu. \tag{2.1}$$

Equation 2.1 describes the change of the Gibbs free energy as a result of the formation of a spherical nucleus of radius R. In the equation, Phase "2" is a state with a nucleus consisting of the crystalline phase and Phase "1" is an entirely liquid state. The change in the free energy of each particle as it goes from liquid to solid is denoted by the chemical potential $\Delta\mu$ in the second term of the equation. Since crystalline phase is more stable, $\Delta\mu$ is negative and the total contribution of the nucleus is calculated by multiplying the change in the chemical potential per particle by the total number of particles in the nucleus (density of the bulk solid times the volume of the solid spherical nucleus). The first term in the equation represents the free energy penalty incurred through the existence of the interface. The interface energy is proportional to the surface area of the nucleus and the constant γ is the free energy surface density, which is positive. Since the two terms that comprise the change in the Gibbs free energy have different signs, the free energy exhibits a positive barrier and monotonically decreases for large values of R.

The free energy of nucleation is plotted as a function of the number of particles in a crystalline cluster in Figure 2.1. Plotting the free energy as a function of the number of solid particles in the nucleus has two advantages, on the one hand, this does not require the cluster to be spherical and, on the other, the size of the critical nucleus can be read directly from the graph.

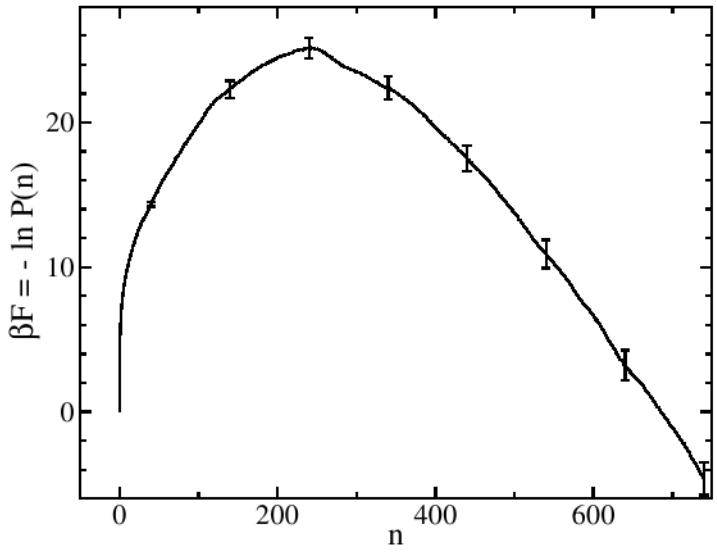

Figure 2.1: The free energy curve of the formation of crystal nuclei in an undercooled Lennard-Jones liquids as calculated in Reference [6].

An interesting conclusion can be drawn from the exitance of such a free energy barrier: very small clusters tend to dissipate and complete crystallization occurs spontaneously only if the system generates a nucleus that is larger than a critical value (in Figure 2.1 the size of the critical nucleus is about 243). The height of the free energy barrier can be calculated from Equation 2.1:

$$\Delta G_{\mathrm{Max}}(R_{\mathrm{Crit}}) = \frac{16\pi\gamma^3}{3\rho_1^2\Delta\mu^2}. \tag{2.2}$$

The value of $\Delta\mu$ depends on the difference between the temperature of the system T and the melting temperature T_m. As $T - T_m$ becomes more negative, so too does $\Delta\mu$, resulting in the reduction the height of the free energy barrier. Conversely, if T approaches the melting temperature the change in the chemical potential vanishes and the potential barrier becomes infinite, corresponding to the observation of liquid-crystal coexistence.

Since the free energy barrier is of the order a $25kT$ and thermal fluctuations are usually of order kT, a pure, undercooled, liquid takes a comparatively long time to spontaneously crystallize. An event that is characterized by such a high free energy barrier is termed a "rare event".

The equations and reasoning presented in this section are all part of the framework of classical nucleation theory. In experimental applications, the presence of impurities, the geometry of containers, and the invalidity of assuming that the crystal nuclei are spherical require either an extension of this simple theory or the microscopic analysis of a simulation. Classical nucleation theory is nonetheless a sufficient starting point and will suffice for the examination of the solid-liquid transition of a pure Lennard-Jones system.

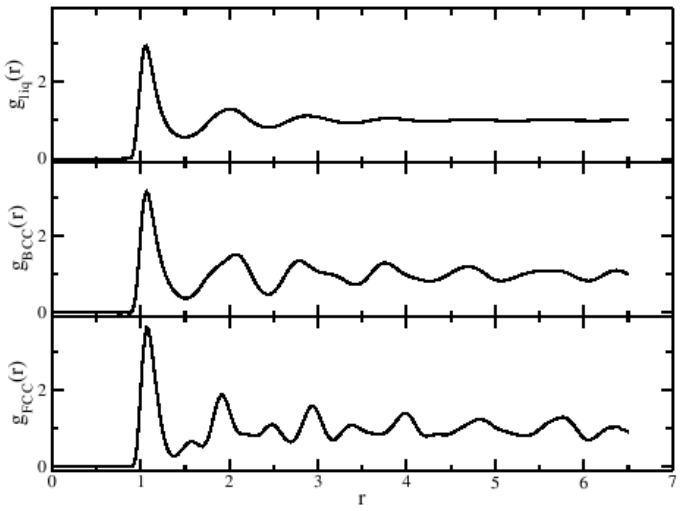

Figure 2.2: The pair correlation functions for a Lennard-Jones liquid, fcc crystal, and bcc crystal close to coexistence [6].

The free energy landscape of a phase transition is mapped as a function of a reaction coordinate which, in this case, is the size of the crystal nuclei that form in an undercooled Lennard-Jones system. The Gibbs free energy is calculated from the probability of finding a cluster consisting of n particles during the course of a simulation,

$$G = -\frac{1}{\beta} \ln[P(n)], \tag{2.3}$$

where $P(n)$ is the probability of finding a cluster of size n.

2.2 Bond Order Parameters and Crystallinity

In order to analyse the phase transitions of the Lennard-Jones system, it is essential to distinguish between the different phases that the system adopts. One of the standard methods was developed by Van Duijneveldt and Frenkel [14] and makes use of the bond order parameters that were introduced by Steinhardt [15]. These bond order parameters have a number of convenient features that make them very effective in the measurement of crystallinity. Using similar nomenclature and conventions as presented in [5], the following bond-order parameters are introduced.

The primary difference between an assortment of solid and liquid particles lies in their geometrical arrangement. This is keenly reflected in the shape of the pair correlation function (Figure 2.2). By developing an algorithm that is sensitive to the specific geometries that particles in an fcc or bcc structure adopt, one can denote particles as part of an fcc or bcc lattice or a disordered liquid. To this end, the following property of the Lennard-Jones particles is defined,

$$q_{lm}(i) := \frac{1}{N_b(i)} \sum_{j=1}^{N_b(i)} Y_{lm}(\hat{r}_{ij}). \tag{2.4}$$

For each particle i the quantity $q_{lm}(i)$ is calculated by examining all particles that are within a prescribed neighbor cutoff distance r_{nc}. $N_b(i)$ is the number of particles that are within the cutoff radius of particle i, and the summation over j is a loop over all of these neighbors.

Y_{lm} are the spherical harmonics which are a set of functions that are orthonormal in an integral sense. The arguments of the spherical functions are the azimuthal angle and the cosine of the polar angle of the vector $\hat{r}_{ij}$, which is the normalized connecting vector between the particles i and j. The index m is circumscribed by the integer index l and assumes the $2l+1$ values $-l, -l+1, ..., 0, ..., l$. The spherical functions are defined by the associated Legendre polynomials,

$$Y_{lm}(\hat{r}_{ij}) := \sqrt{\frac{(2l+1)}{4\pi}\frac{(l-m)!}{(l+m)!}} P^l{}_m(\cos\theta) e^{im\varphi}. \tag{2.5}$$

These bond order parameters inherit the rotational invariance of the spherical harmonics which is advantageous for the examination of crystallinity.

The choice of the cutoff radius r_{nc} for this analysis is very important since the distance between two particles does not play any role in weighting a neighbor's contribution to a particle's q_{lm} value. The difference between the geometries of the phases is most pronounced within the first neighbor shell and that is why the an analysis should be applied only to that region. By examining the radial distribution functions for an fcc and bcc lattice close to coexistence one can determine the distance of the nearest neighbors. The distance of interest corresponds to the first minimum of the pair correlation function of the lattices and can be read off of Figure 2.2.

In Reference [5], a neighbor cutoff of 1.4σ was employed whereas in Reference [6], a value of 1.5σ was used since the latter value is closer to the first minimum of the radial distribution function of the bcc lattice. In Section 5.2.2, it is shown that that the difference between these two parameter values has a minor quantitative but no qualitative effect on the final result.

A number of quantities follow from Identity 2.4. It turns out that using the $q_{6m}(i)$ function is sufficient for a thorough analysis of the fcc, bcc, and liquid geometries. An extended set of order parameters denoted as $w_l(i)$ also exist and they are based on an extension of the $q_{lm}(i)$ parameters and the Wigner-3j symbols. The use of these additional bond order parameters serves to discern between different solid phases more effectively but go beyond the scope of this investigation.

The first quantity to be defined is,

$$q_6(i) := \left[\frac{4\pi}{2l+1} \sum_{m=-l}^{l} |q_{6m}(i)|^2\right]^{\frac{1}{2}}. \tag{2.6}$$

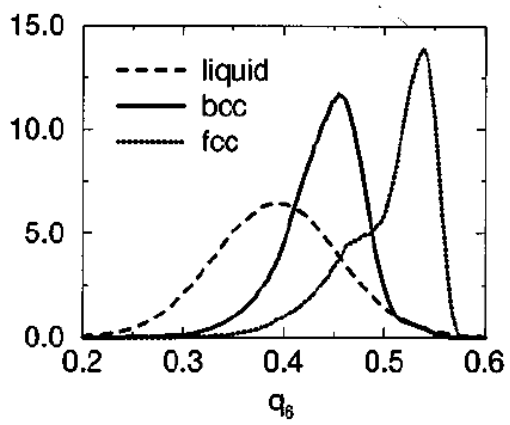

Figure 2.3: Distribution of q_6 order parameter for liquid, fcc and bcc phase close to coexistence [5].

The q_6 parameter is a local bond order parameter and the nature of its distribution is effective in differentiating the three phases that shall be encountered. As can be seen in Figure 2.3, each phase has a unique "thumb print". One is thereby capable of classifying an entire ensemble of particles in either liquid, fcc, or bcc phase.

In order to refine the distributions that are obtained from the different phases and to make statements about individual particles, the normalized vector $\vec{q}_6(i)$ is introduced. Each component of the $\vec{q}_6(i)$ is defined as follows,

$$q_{6m}(i) := \frac{q_{6m}(i)}{\left[\sum_{h=-6}^{6} |q_{6h}(i)|^2\right]^{\frac{1}{2}}}. \tag{2.7}$$

This vector has 13 components $(2l+1)$.

Finally, the dot product between the $\vec{q}_6$ vectors of two particles that are within the neighbor cutoff radius is,

$$\vec{q}_6(i) \cdot \vec{q}_6(j) := \sum_{m=-6}^{6} q_{6m}(i) q_{6m}^*(j). \tag{2.8}$$

The symbol * denotes the complex conjugate of the $\vec{q}_6$ vector component. Since the vectors are normalized, $\vec{q}_6(i) \cdot \vec{q}_6(i)$ is unity.

Any two particles share a bond if they are within the neighbor cutoff of one another and the dot product between their $\vec{q}_6(i)$ vectors serves to classify the bond as either solid or liquid. The graph on the left in Figure 2.4 shows that the separation of phases is even more distinct in the distribution of these dot products. As can be read off the graph, there is a high likelihood that two particles belong to the equilibrated fcc or bcc lattice if the dot product between them is greater than 0.5. Two particles share a "solid" bond if their mutual dot product is above 0.5, otherwise it is a liquid bond. The distribution of the number of solid bonds delivers the final discriminator between particles in solid or liquid environments. In Figure 2.4, the graph on the

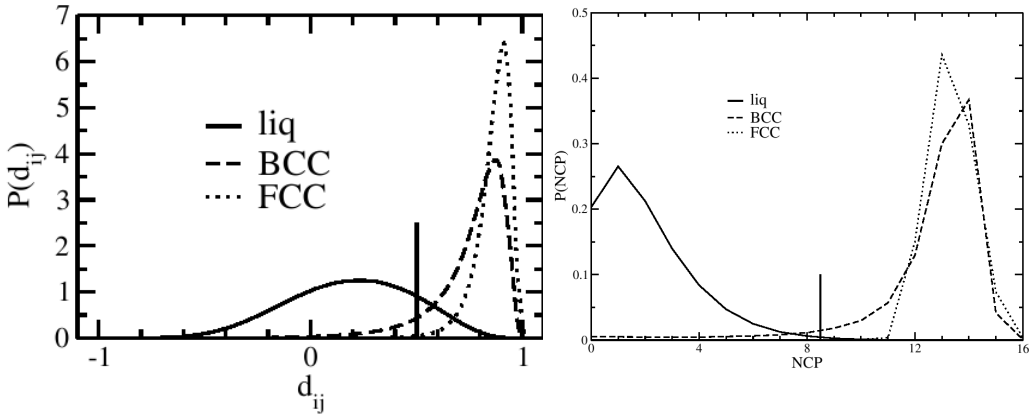

Figure 2.4: Left: distribution of $\vec{q}_6$ dot products. d_{ij} is the value of the dot product between a particle i and a particle j. Right: the number of connected particles (NCP) are plotted on the x-axis, that is, the number of solid bonds that a given particle shares with its neighbors. The probability of finding a particle with that many neighbors is recorded on the y-axis. Both graphs were originally published in Reference [6].

right depicts the distributions of the number of solid bonds of equilibrated systems in different phases at coexistence. A clear criterion has been defined to discern between particles belonging to the liquid phase and particles in an fcc or bcc phase: if the number of solid bonds exceeds 8, a particle is considered to be in the solid phase.

After finding all the particles in the configuration that are solid, one identifies solid clusters in the following manner: two particles are part of the same cluster if they are both solid and within the neighbor cutoff radius of one-another. Thus one is capable of identifying the solid nuclei in an ensemble of particles. In Figure 2.5, a section of a typical configuration that was generated by the simulation is presented (through the use of VMD [16]). The particles that are deemed solid according to the formalism that has been developed are drawn in red whereas the liquid particles surrounding them are yellow.

2.3 Umbrella Sampling

2.3.1 Introduction to the Umbrella Sampling Technique

The simulation of rare events has a pair of challenges associated with it. The first is that the likelihood of crossing a large free energy barrier is, definitionally, very unlikely and this has direct consequences on the length of the simulation runs that are intended to reproduce the event. A computer simulation is usually capable of reproducing the evolution of a physical system from the equivalent of a few pico to nano seconds and therefore an event that occurs every few milliseconds, such as this one, is inaccessible unless special techniques are applied. The second obstacle is related to the first: since there is a high barrier, once the rare event has occurred, the

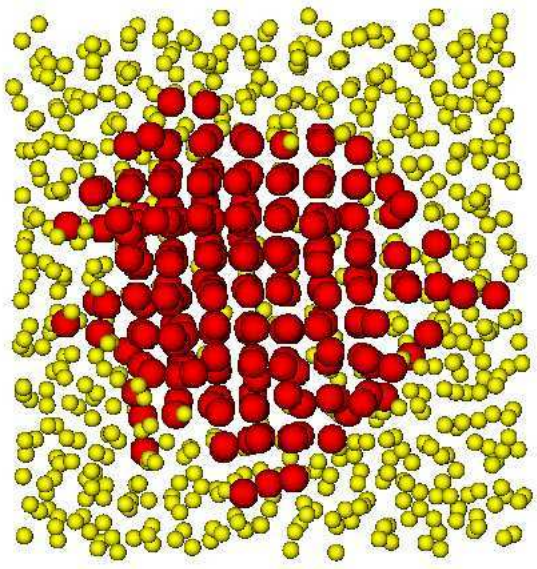

Figure 2.5: A typical configuration that was generated by the simulation. Particles are depicted as either crystalline (red) or liquid (yellow). All particles that have 8 or more solid bonds are considered as part of either an fcc or a bcc structure and it is possible to observe the roughly spherical structure of the nucleus as well as the lattice that it is composed of.

system equilibrates very quickly which means that it is difficult to gather a sufficient amount of data on the phase transition.

In experimental applications the situation is reversed: the timescale for the initiation of a nucleation event is relatively short, the experiments are eminently reproducible and the rate of crystallization can also be measured. It is, however, difficult to infer the microscopic properties of the phase transition with the same degree of detail that computer simulations achieve.

There are number algorithms that address the challenges posed by the rarity of nucleation events and the high rate of crystallization. Since this simulation is modelled after the work done by Lechner [4], the MC-based umbrella sampling technique shall be put to use. A detailed description of the algorithm can be found in Reference [13].

The idea behind umbrella sampling is to assign a small interval of the reaction coordinate to one simulation run and only generate a part of the free energy curve. A simulation that is assigned a segment of the free energy curve is called a window, and by calculating a variety of such windows, one is able to sample the entire interval of the reaction coordinate. Since one can add an arbitrary constant to the Gibbs free energy difference that is computed by each window, the partial curves that are generated can be "pasted" together to form the entire free energy profile.

In Section 2.1, it was alluded that the reaction coordinate of nucleation is the size of the clusters that appear in a window over the course of a simulation run. In order to confine a simulation to a small interval of this reaction coordinate one has

to introduce an artificial biasing potential. To this end, it is necessary to refine the choice of the reaction coordinate as the size of the largest cluster in the configuration. A simple quadratic biasing potential is then introduced,

$$W(N) := \frac{1}{2} k_n \left[N - B\right]^2 . \tag{2.9}$$

In Equation 2.9, N is the size of the largest cluster in the system, B is the cluster value about which the window in question is centered, and k_n is a constant that shall be discussed shortly. After a bond order analysis and a cluster evaluation is applied to a trial configuration, the change in the biasing potential is calculated,

$$\Delta W = W_{\text{New}} - W_{\text{Old}} = \frac{1}{2} k_n \left[N_{\text{New}} - B\right]^2 - \frac{1}{2} k_n \left[N_{\text{Old}} - B\right]^2 . \tag{2.10}$$

In Equation 2.10, N_{New} is the size of the largest cluster in the configuration that has been generated by propagating the configuration once through phase space. B is the cluster size about which the potential is centered and the constant k_n is a parameter that determines how narrow the window is. The denotation of this constant is a reference to the harmonic potential and the subscript is a reminder of the process it is designed to examine.

Now it is possible to evaluate a set of configurations with respect to the free energy landscape and in relation to the window that has generated it. It must be noted that the concerns that arise from simplifying the analysis to only consider the largest clusters that appear in a configuration are justified. The aforementioned simplification is necessary in order to characterize a configuration uniquely. Although systems with large nuclei almost always house only one nucleus, systems with few solid particles tend to have multiple nuclei that are neglected by this method. The artifact that is induced by this shortcoming can be seen clearly in the results presented in Section 5.2 at the beginning of the free energy curve. One obtains the correct curve by simulating the smallest window without a biasing potential and taking all clusters into account as was done by Moroni [6] in order to produce Figure 2.1.

The conventional Monte Carlo acceptance criterion is modified by the following term that is intended to reject configurations that stray too far from the window:

$$P_{\text{Acc}} = \min \left\{ 1, e^{-\beta \Delta U} e^{-\beta \Delta W} \right\} . \tag{2.11}$$

By assigning different bias centers to a number of windows, one is able to calculate the free energy of cluster formation over the entire interval of cluster sizes.

The probability distributions that are generated by the window simulations consist of the true probability of finding a cluster of a given size in the window and the biasing potential that was employed. It is therefore necessary to "unbias" probability distribution $\tilde{P}(N_{\text{Large}})$ that one obtains from the simulation in order to retrieve the desired probability distribution $P(N_{\text{Large}})$:

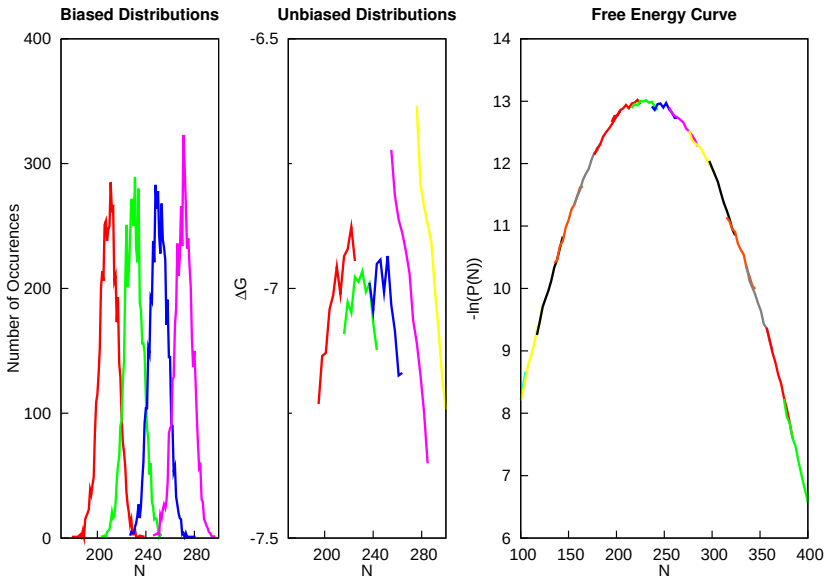

Figure 2.6: First one unbiases the sampled distributions in the leftmost graph and turns them into free energy curves. The resulting curves in the middle are then shifted so that the overlap is maximized to produce the free energy curve on the very right.

$$\begin{aligned} \tilde{P}(N_{\text{Large}}) &= \exp\left[-\frac{k_n}{2}(N_{\text{Large}} - B)^2\right] P(N_{\text{Large}}) \\ &\Rightarrow \\ P(N_{\text{Large}}) &= \exp\left[\frac{k_n}{2}(N_{\text{Large}} - B)^2\right] \tilde{P}(N_{\text{Large}}). \end{aligned} \tag{2.12}$$

Where B is the bias center of the window that generated the distribution. In Figure 2.6, the process of generating a free energy curve from the umbrella sampling method is schematically presented using the data that were gathered from simulations.

2.3.2 Bias Switching

It is thoroughly possible and, in some ways, less problematic to simulate the various windows sequentially, however concurrent simulation makes a very powerful move accessible. Since the biasing potential is a continuous potential (soft windows) and not a discrete wall (hard windows), it is possible that the values of the reaction coordinate of two neighboring windows begin to overlap. One can take advantage of this overlap in the following manner:

- After every modification of the configuration (see Chapter 3), a bias switch move is undertaken by randomly selecting a window and one of its immediate neighbors.

- If N_1 is the size of the largest cluster in the first window, B_1 its bias center, and the quantities N_2 and B_2 are the corresponding values for the second window, then one evaluates,

$$\tilde{W}_{\mathrm{Old}} = (N_1 - B_1)^2 + (N_2 - B_2)^2 \text{ and } \tilde{\mathrm{W}}_{\mathrm{New}} = (\mathrm{N}_1 - \mathrm{B}_2)^2 + (\mathrm{N}_2 - \mathrm{B}_1)^2.$$

- The bias switching move is accepted with a probability,

$$P_{\mathrm{Switch}} = \min\left\{1, \exp\left[\frac{-k_n}{2}(\tilde{W}_{\mathrm{New}} - \tilde{W}_{\mathrm{Old}})\right]\right\}.$$

- If a bias switching move is accepted, then the configuration that was generated by the first window is assigned the second window's bias center and vice versa.

$\Delta\tilde{W} := \tilde{W}_{\mathrm{New}} - \tilde{W}_{\mathrm{Old}}$ measures the effect of switching the biasing centers of two neighboring windows. By performing a random walk in the bias centers, it is possible for ensembles to diffuse through the collection of windows during the course of the simulation, thereby enhancing the sampling that is undertaken. Upon the acceptance of a bias switching move the configurations of the affected windows are entirely decorrelated with respect to the positions of the particles.

2.4 Summary

In this chapter, the $\vec{q}_6$ vector was defined in order to encode the local geometry around each Lennard-Jones particle in the simulation. By considering a particle to be in a solid environment if 8 of the $\vec{q}_6$ dot products between itself and its nearest neighbors exceed 0.5, one is able to distinguish between solid and liquid particles. By conducting a simple cluster search among the solid particles, the crystalline clusters in a configuration can be easily identified. Finally, the umbrella sampling technique overcomes the dual obstacles of the low transition probabilities and the rapid transition rates associated with rare events by assigning to a given simulation a sufficiently small portion of the free energy barrier so that the thermal fluctuations that occur during the course of the run are able to effectively sample the region that it is responsible for.

Chapter 3

The Hybrid Monte Carlo Algorithm

The techniques that were introduced in Chapter 2 to study the nucleation of the Lennard-Jones system are based on the stochastic sampling of configuration space in accordance with the Monte Carlo method. Instead of implementing the Monte Carlo algorithm in which a particle in a given configuration is arbitrarily chosen and then translated in a random direction, which shall henceforth be termed as RDMC (random displacement Monte Carlo), the Hybrid Monte Carlo (HMC) scheme was applied. Much of the following chapter is a summary of the work presented by Mehlig *et al.* [7] on the HMC algorithm.

3.1 Brief Overview of the RDMC Algorithm

The structure and some of the nomenclature of this section was inspired by Daan Frenkel and Berend Smit's treatment of the subject in their book *Understanding Molecular Simulation* [17]. The Metropolis Monte Carlo algorithm itself was first presented in Reference [18].

The thermodynamic properties of a system in equilibrium are encoded in its partition function. For the canonical ensemble, the partition function takes on the following the following form,

$$Q = \frac{1}{C} \int\int d\mathbf{r}^N d\mathbf{p}^N \ \exp\{-\mathcal{H}(\mathbf{r}^N, \mathbf{p}^N)/k_B T\}. \tag{3.1}$$

The variable N denotes the number of particles in the system and thereby defines the dimensionality of the configuration space under consideration. An ensemble defined in 3-space will deliver a partition function that is the result of $6N$ integrations (3 position and 3 momentum coordinates). $\mathcal{H}$ is the Hamiltonian that describes the dynamics of the system, k_B is the Boltzmann factor, T is the temperature of the system (henceforth $\beta = \frac{1}{k_B T}$) and $\mathbf{r}^N$ is an N-dimensional vector, the i-th entry of which corresponds to the position of particle i in the ensemble. Similarly, the N-dimensional vector $\mathbf{p}^N$ describes the momenta of each particle in the system.

Each pairing of $(\mathbf{r}^N, \mathbf{p}^N)$ is considered a point in phase space and is also termed a microstate. For any Hamiltonian each microstate delivers a scalar,

$$\mathcal{H}(\mathbf{r}^N, \mathbf{p}^N) = K + U \tag{3.2}$$

where K is the total kinetic energy of the configuration and U is its total potential energy.

For a system of identical particles, the same value of $\mathcal{H}(\mathbf{r}^N, \mathbf{p}^N)$ is obtained from permutations of the entries of $\mathbf{r}^N$ or $\mathbf{p}^N$ and therefore microstates that are related to each other through the exchange of their entries are indistinguishable. To account for this multiplicity of microstates, the integral in Equation 3.1 requires the constant C. Similarly, a norm exists for the "size" of a microstate derived from the classical approximation of the quantum mechanical ideal gas (the derivation can be found in [19]). As a result of these considerations, $C = h^{3N}N!$ where h is Planck's constant.

A wide variety of systems can be characteriszed by potentials that depend only on the positions of the particles. Such Hamiltonians are said to be decoupled in $\mathbf{r}^N$ and $\mathbf{p}^N$. For,

$$\mathcal{H}(\mathbf{r}^N, \mathbf{p}^N) = \sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m} + U(\mathbf{r}^N), \tag{3.3}$$

(the summation index i runs over all particles in the system and $\mathbf{p}_i^2$ is particle i's momentum squared) one obtains,

$$\begin{aligned}
Q &= \frac{1}{C} \int\int d\mathbf{r}^N d\mathbf{p}^N \ \exp\{-\beta \mathcal{H}(\mathbf{r}^N, \mathbf{p}^N)\} \\
&= \frac{1}{C} \int\int d\mathbf{r}^N d\mathbf{p}^N \ \exp\left\{-\beta \left[\sum_{i=1}^{N} \frac{\mathbf{p}_i^2}{2m} + U(\mathbf{r}^N)\right]\right\} \\
&= \frac{1}{C} \left[\int d\mathbf{p} \ \exp\left\{\frac{-\beta \mathbf{p}^2}{2m}\right\}\right]^N \int d\mathbf{r}^N \ \exp\{-\beta U(\mathbf{r}^N)\} \\
&= \frac{1}{C} \left[\frac{2m\pi}{\beta}\right]^{\frac{3N}{2}} \int d\mathbf{r}^N \ \exp\{-\beta U(\mathbf{r}^N)\},
\end{aligned} \tag{3.4}$$

where $\mathbf{p}$ is a 3-dimensional vector and $\mathbf{p}^2$ is its vector norm squared. The momentum integral in the penultimate line of the calculation above is particularly easy to evaluate analytically over all momentum space and it delivers a constant that is related to the mass of each particle and the inverse temperature β. The partition function for the canonical ensemble becomes,

$$Q = \frac{1}{C'} \int d\mathbf{r}^N \ \exp\{-\beta U(\mathbf{r}^N)\}. \tag{3.5}$$

where C' is the original constant C divided by the contribution of the momentum integral. Although it is rarely possible to calculate the remainder of the partition function analytically, this result is still of great value.

There are a number of useful ensemble averages that depend only on the positions of the particles and they are expressed in this formalism as,

$$\langle A \rangle = \frac{\int d\mathbf{r}^N \ \exp\{-\beta U(\mathbf{r}^N)\} A(\mathbf{r}^N)}{\int d\mathbf{r}^N \ \exp\{-\beta U(\mathbf{r}^N)\}}. \tag{3.6}$$

From Equation (3.6) one can infer that,

$$\mathcal{P}(\mathbf{r}^N) = \frac{\exp\{-\beta U(\mathbf{r}^N)\}}{\int d\mathbf{r}^N \ \exp\{-\beta U(\mathbf{r}^N)\}} \tag{3.7}$$

acts as the probability density of the canonical ensemble. Since the denominator of the aforementioned term cannot be calculated analytically, a different method must be applied to obtain further results. By reducing the partition function of the NVT ensemble to the probability density in Equation 3.7, only the positions of the particles under consideration are of importance and therefore a configuration can fully described by a $3N$-dimensional vector. Such vectors shall be henceforth termed x and x'.

The RDMC algorithm is a means by which one can sample the Boltzmann distribution directly evaluating the Equations 3.5 and 3.6 provided that the "ergodic hypothesis" holds and that detailed balance is preserved. If these two conditions are fulfilled, the RDMC algorithm generates a stationary Markov chain of configurations that corresponds to the probability density in Equation 3.7 and through the analysis of the ensuing configurations, accurate estimates of the thermodynamic averages of in Equation 3.6 can be made.

The "ergodic hypothesis", that is either assumed or proven for many of the systems that undergo statistical analysis, states that, "every accessable point in configuration space can be reached in a finite number of Monte Carlo steps from any other point" [17]. It is important that this condition hold in order to guarantee that the calculated ensemble averages of the algorithm converge to those of the system.

In order to generate a sequence of configurations that sample the Boltzman distribution, the condition of detailed balance is imposed on the trial configurations x' that are generated stochastically from a configuration x. To observe detailed balance, the probability of the occurrence of a configuration x, called $\mathcal{P}(x)$, times the probability of x transitioning to a state x', called $\pi(x \to x')$ must equal $\mathcal{P}(x')$ times $\pi(x' \to x)$. Mathematically the condition is written as,

$$\mathcal{P}(x)\pi(x \to x') = \mathcal{P}(x')\pi(x' \to x). \tag{3.8}$$

Detailed balance is the statistical formulation of the statement that two configurations belonging to phase space have equal and opposite transition currents. Simple algebraic manipulation delivers,

$$\frac{\pi(x \to x')}{\pi(x' \to x)} = \frac{\mathcal{P}(x')}{\mathcal{P}(x)}. \tag{3.9}$$

The transition probability $\pi(x \to x')$ can be decomposed into two new probabilities: the probability of generating x' given x, $P_{\text{Gen}}(x \to x')$, and the probability of

accepting x' in favor of x, $P_{\mathrm{Acc}}(x \to x')$. The RDMC move, as well as many other moves that have been developed for Monte Carlo simulations are constructed in such a manner that $P_{\mathrm{Gen}}(x \to x') = P_{\mathrm{Gen}}(x' \to x)$, that is to say, reversing the move that led to x' from x is as likely as generating the move that leads from x to x'.

Since the quotient of the probabilities of the two states appears on the right hand side of Equation 3.9 only the relative probabilities of the two states are of importance. By using the identities and prescriptions of the previous paragraph as well as inserting Equation 3.7 into Equation 3.9, one obtains:

$$\begin{aligned} \frac{P_{\mathrm{Gen}}(x \to x')P_{\mathrm{Acc}}(x \to x')}{P_{\mathrm{Gen}}(x' \to x)P_{\mathrm{Acc}}(x' \to x)} &= \frac{\mathcal{P}(x')}{\mathcal{P}(x)} \\ \frac{P_{\mathrm{Acc}}(x \to x')}{P_{\mathrm{Acc}}(x' \to x)} &= e^{-\beta \Delta U}. \end{aligned} \tag{3.10}$$

Where $\Delta U = U(x') - U(x)$ is the potential energy difference between the configurations x' and x. In order to finally obtain the acceptance probability of a generated configuration one needs to examine the relation

$$P_{\mathrm{Acc}}(x \to x') = e^{-\beta \Delta U} P_{\mathrm{Acc}}(x' \to x). \tag{3.11}$$

Although there are many acceptance criteria that fulfill the condition above, one of the most prolific is the "Metropolis Criterion",

$$P_{\mathrm{Acc}} = \min\left\{1, e^{-\beta \Delta U}\right\}. \tag{3.12}$$

The RDMC method is therefore implemented in the following manner:

1. Begin with a configuration x and calculate its potential energy $U(x)$.
2. Generate a new configuration x' from x by selecting a particle at random and displacing it in an arbitrary direction. This is a possible move that observes the condition $P_{\mathrm{Gen}}(x \to x') = P_{\mathrm{Gen}}(x' \to x)$.
3. Evaluate $U(x')$ and accept x' with a probability of $P_{\mathrm{Acc}} = \min\left\{1, e^{-\beta \Delta U}\right\}$.
4. In case of an acceptance of the configuration, x' becomes the new configuration x. In case of a rejection the system remains at x.
5. Repeat steps 1 to 5 until the configuration space has been sufficiently sampled to produce accurate thermodynamic averages.

It must be stressed that any method of changing the positions of the configuration is acceptable, provided that the probability of generating new configurations preserves detailed balance. In Figure 3.1, a simple depiction of the RDMC algorithm is presented. The chosen particle is translated to some point within a square with sides of length $2L$. The average acceptance rate of trial configurations decreases as the L parameter is increased because the variation in the potential energy tends to

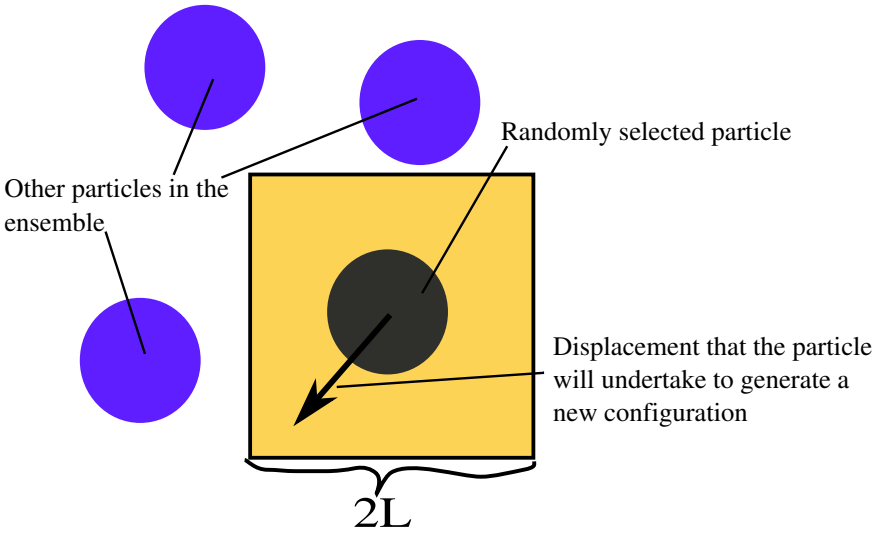

Figure 3.1: A two-dimensional depiction of a RDMC move. The gray particle has been randomly selected from the ensemble to be displaced, the shaded region about the particle represents all of the positions that it could be translated to, and the arrow is the randomly generated displacement that it will undertake in order to produce a trial configuration.

grow with large displacements thereby making the acceptance of trial configurations less likely. A rule of thumb for computational physicists is to tune the displacement parameter L in such a manner as to obtain a $30\% - 50\%$ acceptance rate of trail configurations, since this tends to optimize the rate at which configurations decorrelate from one another.

3.2 Motivation for a New Monte Carlo Move

Attempts to apply multi-particle RDMC moves to the MC algorithm have been plagued with appallingly low accept rates. Recently, efforts have been made to develop an alternative Monte Carlo move that changes the positions of many particles at once. A suggestion was made by Duane *et. al.* [20] to "guide the Monte Carlo simulation" through the integration of the Hamiltonian equations in the simulation of lattice gauge theories and later Mehlig's group applied this concept to condensed matter systems [7]. If it can be shown that the generation of a trial configuration through the integration of the Hamiltonian equations of motion preserves detailed balance, then such an algorithm is a valid Monte Carlo move. It is then possible to formulate a global move that fits into the Monte Carlo scheme.

3.3 A New Trial Move

The basic idea of the HMC algorithm for condensed matter systems is to employ a conventional Molecular Dynamics algorithm that is based on integrating the equa-

tions of motion of the system. By discretizing the time axis of the system, the equations of motion can be integrated approximatively and it is well known and stipulated, that in the limit of vanishingly small time discretization, δt, that the total energy of the system is perfectly conserved by the algorithm and that the integration becomes exact. When conducting a Molecular Dynamics (MD) simulation, both the choice of the integrator of the equations of motion and the choice of the δt parameter are chosen so that the Hamiltonian is almost perfectly conserved. In this case however, δt is adjusted so as to deliver varying values of the total energy as a result of the numerical error caused by the algorithm. In Reference [7], it is shown that configurations that are propageted in such a fashion for a few MD steps generate appropriate candidates for the Metropolis Monte Carlo acceptance criterion if random velocities are drawn from a Maxwell-Boltzman distribution after every sequence of integrations. A single HMC move consists of the following steps:

1. Calculate the forces acting on all particles in the current configuration.

2. Integrate the equations of motion using a time reversible and area preserving algorithm, (e.g. Velocity Verlet) with a discretization step δt.

3. Repeat the previous two steps a number of times. The number of such repetitions will be referred to as N_{MD}.

4. Accept or reject the ensuing configuration with an acceptance rate of $P_{\text{Acc}} = min\{1, e^{-\beta\delta\mathcal{H}}\}$, whereby $\delta\mathcal{H}$ is the difference between the total energy of the trail configuration and the original one.

5. In case of an acceptance, the trial configuration x' becomes the new configuration x.

6. In any case, assign new velocities that are drawn from the Maxwell-Boltzmann distribution $\frac{1}{N}e^{-\frac{\beta p^2}{2m}}$. This step delivers a velocity distribution that is compatible with the temperature that the ensemble is simulated at and propagates the system along a new trajectory in phase space.

The final step in the "recipe", in which new velocities are drawn for all particles after each HMC move, guarantees that $P_{\text{Gen}}(x \to x') = P_{\text{Gen}}(x' \to x)$.

Thus, instead of using an arbitrary displacement of a single particle, the HMC algorithm propagates the entire configuration according to the equations of motion. There are two parameters that influence the acceptance rate of trial configurations: the first is the discretization step δt and the second is the parameter N_{MD}.

3.4 Efficiency Considerations

In this section an attempt will be made to optimize the two parameters δt and N_{MD} that influence the acceptance rate of the HMC algorithm.

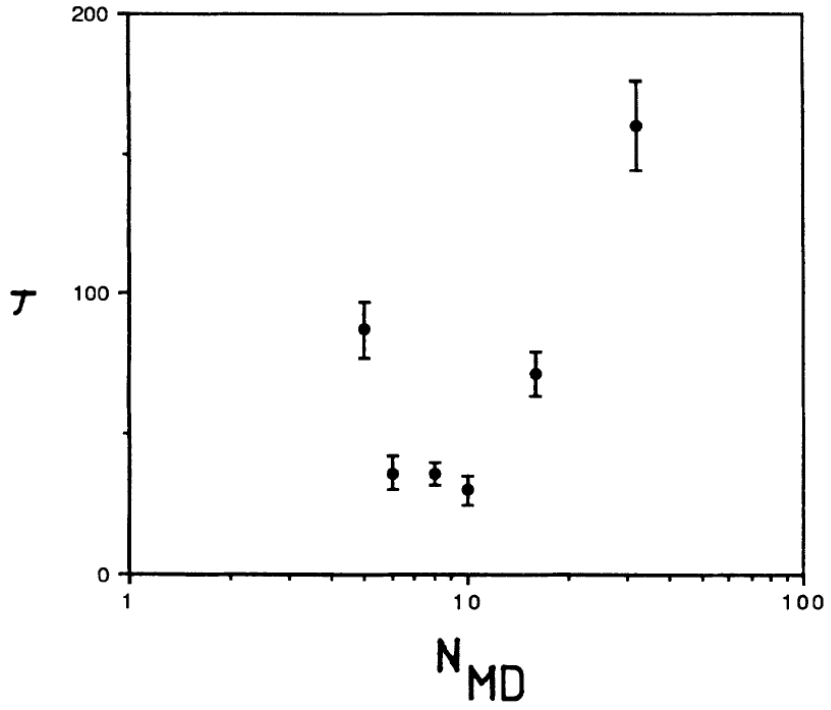

Figure 3.2: Autocorrelation time of the potential energy of a Lennard-Jones system vs the number of MD steps employed per HMC step [7] (δt is adjusted in such a manner as to keep $t_{HMC} = N_{MD}\delta t = 1$). τ is given in the number of HMC steps. As can be seen, the shortest correlation time occurs at $N_{MD} = 10$ and $\delta t = 0.1$.

3.4.1 Optimizing trajectory lengths

After introducing the HMC algorithm, the remainder of Reference [7] is dedicated to determining the optimal values of δt and N_{MD} for a Lennard-Jones system and a treatment of the Ferrenberg-Swendsen extrapolation [21, 22] (the latter topic is irrelevant for this discussion of the HMC algorithm). The efficiency of a Monte Carlo scheme can be evaluated quantitatively through the analysis of the correlation functions of the ensemble averages that the algorithm calculates.

Mehlig *et. al.* analyzed a Lennard-Jones system with 256 particles. The unit of energy and the unit of length were set to the parameters 48ϵ and σ in the Lennard-Jones potential. k_B was set to unity. The simulations were conducted close to the coexistence curve at a reduced temperature of $T = 0.72$ and reduced density of $\rho = 0.83$. An interaction cutoff $r_c = 2.5$ was employed and the ensuing error in the potential calculation was not corrected for. A leap-frog algorithm was implemented in order to integrate the equations of motion.

The principle behind the ensuing analysis is that the combination of simulation parameters that induces the shortest autocorrelation time of the potential energy (the main thermodynamic quantity that the RDMC and HMC algorithm calculates) represents the most efficient means of sampling the canonical ensemble.

The analysis begins by defining $t_{HMC} = N_{MD}\delta t$ as the length of the Molec-

ular Dynamics trajectory that the system is propagated along to produce a trial move. It is known that the discretization error of any integrator rises both with the discretization size, δt, and the number of integration steps, N_{MD}, applied to the system. The trajectory t_{HMC} was kept at a length of 1 and various values of N_{MD} (with the appropriate δt) were used as the parameters for an NVT simulation for which the autocorrelation time of the potential energy was calculated and plotted in Figure 3.2 as a function of N_{MD}.

Another analysis was initiated by setting $N_{MD} = 10$ and varying δt in order to find the shortest autocorrelation time of the potential energy in Figure 3.3.

It is surprising that the optimal values $N_{MD} = 10$ and $\delta t = 0.1$ deliver an acceptance probability of around 70% (see Figure 3.4), which is a great deal higher than the usual 50% acceptance rate to which RDMC algorithms are tuned to.

3.5 Sampling Other Ensembles

If one finds the need to conduct a simulation using other ensembles, the same procedures and algorithms can be appended to the HMC algorithm as those used with the RDMC scheme. Thus, in order to simulate the NPT ensemble, the same volume move can be implemented as the one described by Frenkel and Smit in [17]. An MC barostat in the Monte Carlo algorithm consists of suggesting a random expansion or contraction of the system that is implemented by rescaling all lengths of the system (i.e. the sides of the simulation box and all the positions of the particles) and evaluating the quantity,

$$\Delta\varphi = \Delta U + P\Delta V - \frac{N+1}{\beta}\ln\left(\frac{V'}{V}\right), \tag{3.13}$$

and accepting the trial configuration with probability:

$$P_{\text{Acc}} = \min\left\{1, \exp^{-\beta\Delta\varphi}\right\}. \tag{3.14}$$

The suitability of the trial configuration depends on the change in the potential energy as a result of rescaling all particle positions (ΔU), the work done to or by the system to change the volume ($P\Delta V$), and the change in the entropy as a result of the variation of the volume ($\beta^{-1}[N+1]\ln[V'/V]$). It must be noted that in order to guarantee that the ensuing Markov chain is symmetric, a change in the volume must be performed randomly with a preset probability P_{vmove}.

Frenkel and Smit [17] proceed to describe the distribution with which the random walk in $\log[length]$ space must be performed in order to obtain the correct thermodynamic averages. When a volume move is conducted, all distances are scaled by a factor, $\frac{L'}{L}$, drawn from the distribution,

$$\frac{L'}{L} = \exp\left\{\frac{(\mathcal{X} - 0.5)M_{\text{Size}}}{3}\right\}. \tag{3.15}$$

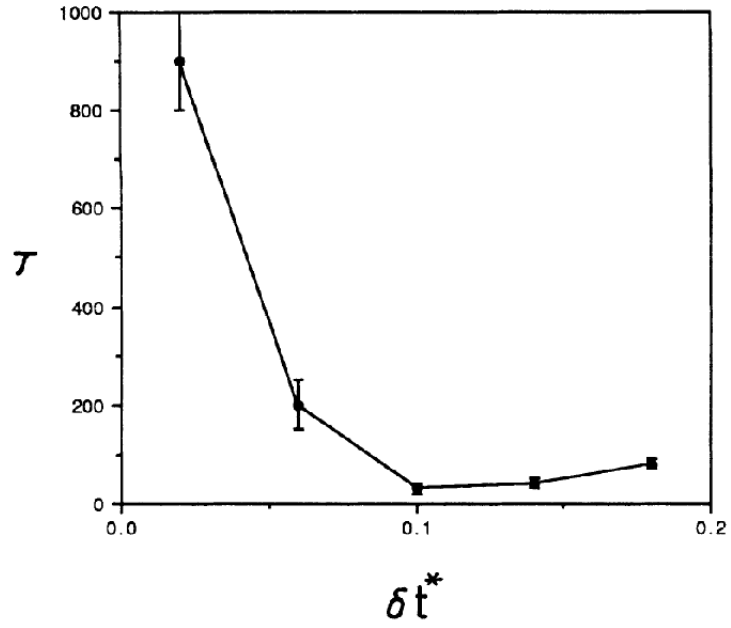

Figure 3.3: Autocorrelation time of the potential energy of a Lennard-Jones system vs the discretization size δt (N_{MD} is kept at 10) [7]. The shortest correlation time occurs at $\delta t = 0.1$.

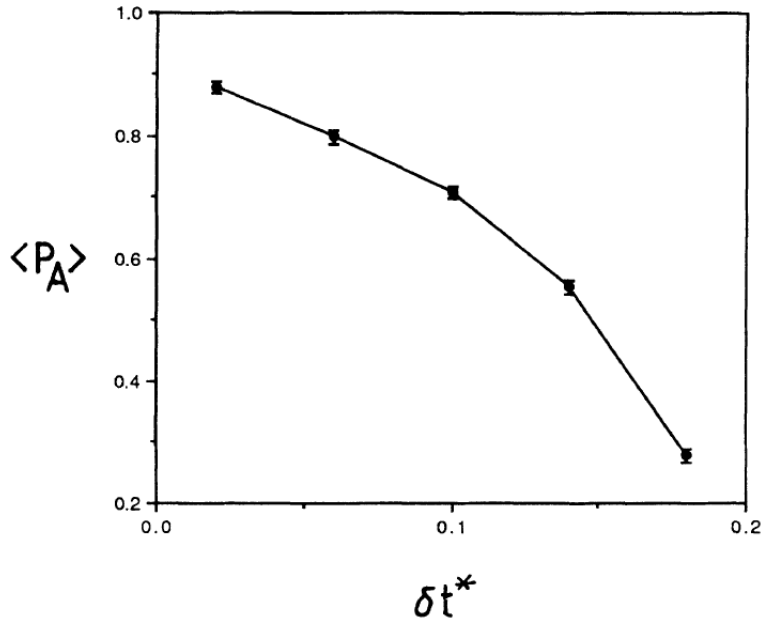

Figure 3.4: HMC acceptance rates as a function of the discretization step, δt, for $N_{MD} = 10$. The shortest autocorrelation time of the potential energy is obtained for $\delta t = 0.1$ corresponding to an acceptance rate of 70%.

$\mathcal{X}$ is a random variable drawn from the uniform distribution $[0, 1]$. The parameter M_{Size} determines the average size of the suggested rescaling and directly affects the acceptance rate of volume moves.

3.6 Summary

Two perceived advantages are derived from the use of HMC algorithm. The first is that global moves are conducted by the HMC algorithm, the acceptance rates of which can be tuned through the manipulation of the discretization parameter δt (the number of MD steps per HMC step will be henceforth set to $N_{MD} = 10$). Second, the HMC algorithm employs virtually all of the functions required by a Molecular Dynamics simulation whilst still being a Monte Carlo algorithm. In a certain sense, tracking the GPGPU's performance of the HMC algorithm therefore provides insights into the device's capability in conducting both MC and Molecular Dynamics simulations simultaneously.

Chapter 4

Implementation of CUDA Functions

The purpose of this chapter is to discuss the concrete implementations of the algorithms that were used. By detailing the structure of the functions that were written and the CUDA specific measures that had to be undertaken it is hoped that reader will have a better sense as to how the GPGPU was able to achieve the speedup factors that are presented in Chapter 5.

A familiarity with the CUDA guide written by NVIDIA presupposed. The guide is available for download from their website [23] and the CUDA specific commands used in the code are CUDAMalloc, CUDAMemcpy, and syncthreads(). Additional commands were unnecessary but may be instrumental in higher level performance enhancement.

The architecture of a CUDA device specifically aims to optimize the essential features of effective parallel computing. Specifically, the degree of parallelization and the efficiency of thread communication are oftentimes the performance bottlenecks of parallel applications. The CUDA programme addresses these limitations in the following manner:

- A GPGPU inherently features an architecture that is predisposed towards parallelism. Whilst a conventional cluster offers around 100 nodes, a 1.x compute capability GPGPU can launch a maximum of 512 copies of kernel code in a single block. Since each block is processed by one multiprocessor and there are 16 such multiprocessors on a Tesla C870 model, a CUDA program can be coded with around 8192 processes (called "threads") in mind. Although this is by no means an upper limit to the size of the execution grid, it might be more sensible to use fewer threads per thread block but compile more blocks for execution. It is also important to note that roughly $32 \times 16 = 512$ (warp size $\times$ number of multiprocessors on the GPGPU) processes will be calculated concurrently and the remaining threads will be queued, waiting for execution.

- Functions are called from the CPU (henceforth referred to as the "Host") during the execution of a program with block sizes and grid dimensions being variable

arguments in the function call. It is therefore possible to design the optimum execution grid based on the GPGPU employed on a function-to-function basis.

- The global memory of the GPGPU can be accessed by all threads during the entire duration of a function call and remains intact even after it has completed execution. Thus the data stored on the GPGPU during the lifetime of the application can be accessed and manipulated by a variety of functions featuring mutually independent execution grids. The shared memory, on the other hand, is a feature that is an extremely effective means of high speed data exchange between the threads of a block during a function call. The shared memory is allocated when a function is called, and its size is variable between separate function calls.

As a result, the user is able to slice the data that is to be computed in whichever way is deemed most effective by means of the longevity of global memory and variable execution grids, has access to features that specifically allow for a high degree of communication between threads (shared memory), and through the use of the cudaMemcpy command is able to conduct computations, compile data, and prepare inputs that are to be uploaded to the GPGPU, or take advantage of CPU exclusive features on the Host.

During the conception of the project (much credit must be attributed to J.A. van Meel both in terms of personal conversations and professional accomplishments [3]), it was deemed most sensible construct this umbrella sampling experiment in the following manner: each window consists of an NPT Monte Carlo system which is propagated once through phase space via either an HMC step or by a volume move by the graphics card (see Chapter 3). After subjecting each "Simulation Box" to a bond order analysis (see Chapter 2), the cluster data is collected, a bias switch is suggested, and then the propagation begins anew.

In order to ascertain which of the functions required tweaking and what the optimal configuration of the simulation parameters was, the CUDA profiler was employed. It can be downloaded from the NVIDIA CUDA website [10]. As is to be expected, the CUDA visual profiler output (Figure 4.1) suggests that the calculation of the Lennard-Jones forces (Forces) and the collection of the data to evaluate the local geometry around each particle (Fill Neighbour Lists) are by far the most expensive functions in the simulation.

The functions that calculate the pair forces between the particles and collect the neighboring geometry will be discussed in Section 4.1 (refer to Section 2.2 for more on the nature of the geometric analysis). The integration function (Integrate), which applies the Velocity Verlet algorithm on each particle, the bond order analysis of each particle (GPU Calculate Q6M and Evaluate Solidity), the assignment of which cell a particle belongs to (Assign Cells), the sampling of new velocities after each Hybrid Monte Carlo move (Sample Velocities), and the functions responsible for restoring configurations in case of a rejection of a Hybrid Monte Carlo or window sampling move (Backup and Revert Configs) are functions that are "embarrassingly parallelizable" and are the subject of Section 4.2. The routines responsible

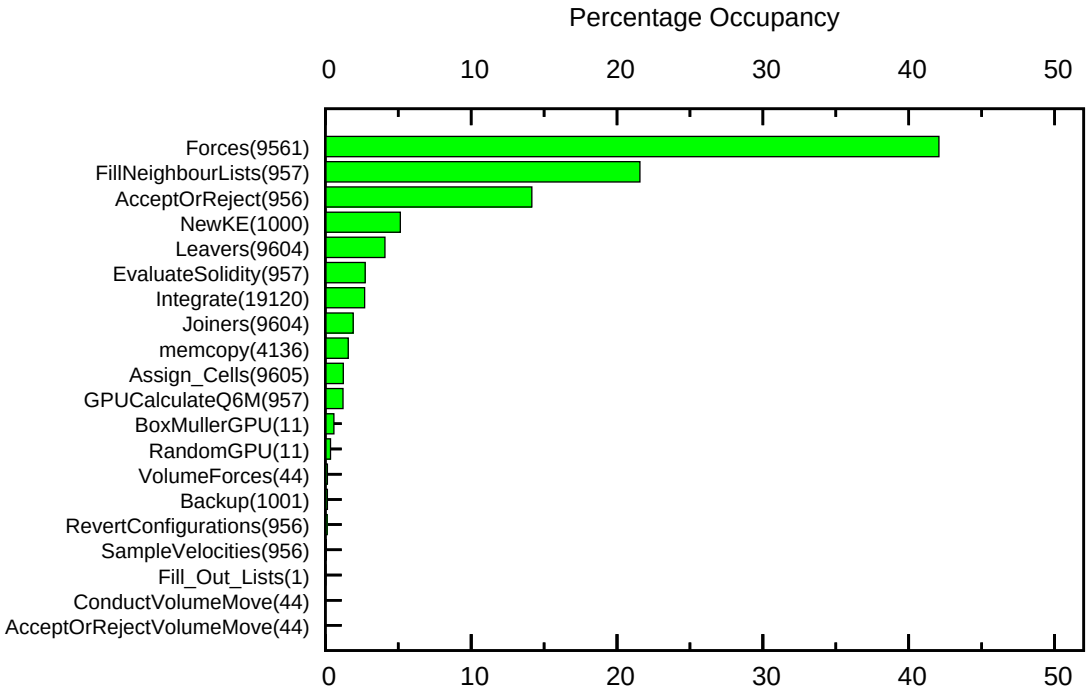

Figure 4.1: CUDA Visual Profiler summary of the functions employed in the simulation versus the percentage of the total time that was required by each function. The values that are in parenthesis next to the function names are the number of times the function was called.

for the maintenence of the cell lists employed in the program are yet another class of functions (Joiners and Leavers) that are attended to in Section 4.3.

4.1 Force-Type Functions

Although cell lists are a very effective means of reducing the computational time of multiparticle simulations with cutoff radii in-and-of themselves, the CUDA architecture naturally calculates the particle interactions very efficiently if cell lists are employed.

By decomposing each Simulation Box into cubic cells with a base length that is at least as long as the cutoff radius, all particles that will interact with the inhabitants of a given cell will be contained in either the cell itself or in the cell's 26 adjacent neighbors. After assigning a global index to each cell, an execution grid is launched in which one cell is computed by one thread block (see Figure 4.2). Finally, every interaction throughout the system is accounted for by referencing an array that has calculated which cells are adjoined to the thread block's cell. Furthermore, if one decomposes a Simulation Box into more than 64 (4×4×4) cells, one can implement the nearest image convention by translating all particles of a neighbor cell if its interaction partner is located on the opposite side of the Simulation Box,

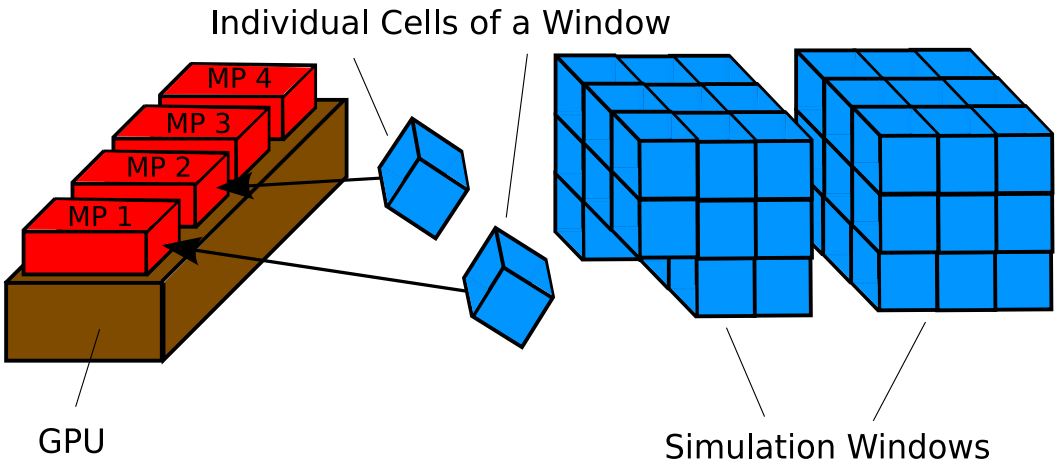

Figure 4.2: The symbolic decomposition of each NPT ensemble into a series of cells that are processed concurrently by the GPGPU's various multi-processors (each of which is designated as an MP in the figure) during the calculation of the Lennard-Jones forces.

making it unnecessary to ascertain as to whether or not a given displacement vector's component is greater than half the box length.

4.1.1 Implementation of the Force Function

This implementation of the Force function interprets a cell as one block in the grid that is launched and a thread as a single entry in the cell. Each thread identifies the cell it belongs to based on its "block Id" and then loads the particle data (position and particle Id) in the cell list corresponding to its own "thread id". For the rest of the function call, the thread is responsible for this particle. Then, each thread proceeds to load its particle's position and id into shared memory. After thread synchronization has taken place, each thread references the data in the shared memory sequentially in order to calculate the force and potential acting on its particle as a result of the particles cohabiting its cell. Each particle's effect is incrementally added to temporary local variables. Care must be taken to avoid singularities resulting from self interactions and although this would be a simple matter on a conventional CPU, the unavoidable use of conditional code has ramifications that will be discussed in Section 4.1.2.

After the threads are synchronized once more, the contents of a neighboring cell is fetched by each thread and saved into the block's shared memory. The threads are synchronized again to ensure that all the data is loaded and then each thread calculates its particle's interactions with all other particles stored in the shared memory of the thread's block until the entire cell has been processed. By iterating over all neighboring cells every relevant interaction is accounted for. Finally, each thread proceeds to save the compiled forces in the global memory slots dedicated to the forces acting on the particle.

In the following pages, a number of figures shall detail the exact structure of

the force function in order to illustrate the concepts discussed in this paragraph. Furthermore, a number of incarnations of the this function will be presented in order to develop CUDA specific concepts. All versions of the force function have the code presented in Figures 4.3a and 4.3b in common. The remainder of the function will be elaborated upon in subsequent figures.

```
__global__ void ForceFunction(CList *CellList, Particle *Molecules,int
    *ListOfCellNeighbours){

//The arguments of the function are the arrays CellList, Molecules and
    ListOfCellNeighbours, thereby making them accessable to all threads
    . The CellList array has all particle's positions and global id's
    saved for each cell in the simulation box. Molecule contains all
    particle information. ListOfCellNeighbours contains the cell Ids of
     all cells that are adjacent to a given cell.


//Each thread's cell number and position in the cell is determined.
  int MyCellnumber=blockIdx.x;
  int myEntryNumber=threadIdx.x;

//The index to which the thread's cell information is assigned to in
    the CellList array is set.
  int MyCellListIndex=....;

//The number of particles inhabiting the current thread's cell is
    loaded.
  int MyCellPopultaion=....;

//An auxiliary variable.
  double r2;

//An array that resides in shared memory and is accessable to all
    threads within a block.
  __shared__ CList Partners[];

//Loading the particle that the thread is responsible for into local
    registry memory.
  vec3 myparticlePosition=CellList[MyCellListIndex].Pos;

//Loading particle Id and declaring auxiliary variable. NOTE: The vec3
    type is a struct that has the float members .x,.y, and .z
  int myparticleId=CellList[MyCellListIndex].Id;
  vec3 DisplacementVec;

//First one loads each particle in the current cell into shared memory
    so that all interactions between particles within the cell can be
    computed.
  Partners[threadIdx.x].Pos=myParticlePosition;

 //The threads in the cell block are synchronized in order to ensure
    the data is loaded.
    __synchthreads();
```

Figure 4.3a: Basic structure of the force function.

```
//For all inhabitants in the cell...
  for (i=0;i<MyCellPopulation;i++){
//...except for the particle itself...
    if(i!=threadIdx.x){
//...displacements are calculated.
      DisplecementVec.x= Partners[i].Pos.x-myParticlePosition.x;
      DisplecementVec.y= Partners[i].Pos.y-myParticlePosition.y;
      DisplecementVec.z= Partners[i].Pos.z-myParticlePosition.z;

//The distance between two particles is calculated...
      r2=GetNorm(DisplacementVec);
//...and if the particles are within the interaction cutoff,
      if(r2<LJCUTOFFsquared){
//Lennard-Jones forces and potentials are calculated.
        GetForcesAndPotential(DisplacementVec);
      }
    }
  }

//Synchronizing all threads to ensure calculations are completed before
    considering the adjacent cells.
__synchthreads();

//Subsequent Code
 ....
 ....
 ....
}
```

Figure 4.3b: Continuation of the basic structure of the force function.

4.1.2 CUDA Specific Programming

During the analysis of the application's performance, three GPGPU specific effects were discovered. The first phenomenon involves the access that threads have to global memory. The two remaining effects are related to the pitfalls of thread branching and the use of conditional code in CUDA.

Global Memory Access

Usually, when a particle calculates the interaction force and potential with another particle, the data is saved both for the particle in question and its interaction partner. As a result only half of the particles need to be considered. Figure 4.4 depicts such an intuitive program structure. The cell list algorithm is implemented by only considering half of the neighboring cells, and whenever an interaction partner of a particle in a cell is identified, the effect of the interaction is also recorded for the interaction partner.

On a GPGPU application such a programming structure does not function. Although one could try to implement the code in Figure 4.4 by having each cell calculate interactions for itself and half of its neighbors, it is impossible for multiple of threads to modify the same global memory slots simultaneously. This is precisely what would happen if one particle's effect on an interaction neighbor would be calculated by two threads at once, since both threads would be instructed to increment the forces acting on this particle at the same time. A memory write conflict would occur and the data would be corrupted. It is also cumbersome (due to memory constraints) for a thread to remember the effect that each particle has on it and to compile the data at a later point in time. As a result, each cell has to consider all of its neighbors. The force and potential calculations conducted in this experiment take on the form featured in Figure 4.5.

One can see that the GPGPU version of the code has an inherent factor 2 more calculations that need to be conducted as compared to the CPU version of the code.

Note: If one insists on using a force function without the disadvantageous factor 2 more calculations, an interaction matrix model might be used: by launching an execution grid that is meant to calculate the matrix elements of the interaction matrix [F_{ij}] (each element of this matrix represents the forces acting on particle i due to particle j) one can distribute the elements that need to be calculated across the execution grid. Since the interaction matrix is antisymmetric and the diagonal elements are all zero, only the upper triangle of the matrix needs to be computed. Afterwards, the lower triangle is filled out appropriately. A separate summation step could then be used to calculate the net force acting on each particle.

```
__global__ void ForceFunction(CList *CellList, Particle *Molecules, int
     *ListOfCellNeighbours){
  double Force;
  int MyNeighboursId;
  ...
//The beginning of this function is depicted in the previous two
    figures. The code in this figure begins when the interaction with
    neighboring cells is considered.

//Synchronizing threads before loading neighbor cell data.
  __synchthreads();

//Only loop over 13 of the 26 neighboring cells and attempt to modify
    two particles at once in the following lines.
  for (cell=0;cell<CellNeighbours/2;cell++){
    NeighbourCellIndex=ListOfCellNeighbours[MyCellListIndex][cell];
    Partners[threadIdx.x].Pos=CellList[NeighbourCellIndex].Pos
    NeighbouringCellPopulation=....;
    __synchthreads();

    for (i=0;i<NeighbouringCellPopulation;i++){
      DisplecementVec.x= Partners[i].Pos.x-myParticlePosition.x;
      DisplecementVec.y= Partners[i].Pos.y-myParticlePosition.y;
      DisplecementVec.z= Partners[i].Pos.z-myParticlePosition.z;

//First, make note which particle is being considered...
      MyNeighboursId=....; r2=GetNorm(DisplacementVec);
//...then check to see if the particles are interaction partners...
      if(r2<LJCUTOFFsquared){
        Force=GetForces(DisplacementVec);
//...increment the forces of the thread's particle...
        Molecule[myparticleId].For.x=Molecule[myparticleId].For.x +
            Force*DisplacementVec.x;
        Molecule[myparticleId].For.y=Molecule[myparticleId].For.y +
            Force*DisplacementVec.y;
        Molecule[myparticleId].For.z=Molecule[myparticleId].For.z +
            Force*DisplacementVec.z;
//...and those of its interaction partner.
        Molecule[MyNeighboursId].For.x=Molecule[MyNeighboursId].For.x -
             Force*DisplacementVec.x;
        Molecule[MyNeighboursId].For.y=Molecule[MyNeighboursId].For.y -
             Force*DisplacementVec.y;
        Molecule[MyNeighboursId].For.z=Molecule[MyNeighboursId].For.z -
             Force*DisplacementVec.z;
        Molecule[myparticleId].Potential=Molecule[myparticleId].
            Potential + GetPot(DisplacementVec);
      }
    }
    __syncthreads();
}
```

Figure 4.4: A force function for which memory-write conflicts occur.

```
__global__ void ForceFunction(CList *CellList, Particle *Molecules,int
    *ListOfCellNeighbours){

  double Force,mypotential;
  vec3 MyForces;
       ....
//Again, the code only changes in the consideration of the neighboring
    cells.

  __synchthreads();
 //One is forced to loop over all adjacent cells...
  for (cell=0;cell<CellNeighbours;cell++){
    NeighbourCellIndex=....;
    InteractionPartners[threadIdx.x].Pos=CellList[NeighbourCellIndex].
        Pos
    NeighbourCellIndex=ListOfCellNeighbours[MyCellListIndex][cell];

    for (i=0;i<NeighbouringCellPopulation;i++){
      DisplecementVec.x= InteractionPartners[i].Pos.x-
          myParticlePosition.x;
      DisplecementVec.y= InteractionPartners[i].Pos.y-
          myParticlePosition.y;
      DisplecementVec.z= InteractionPartners[i].Pos.z-
          myParticlePosition.z;
      MyNeighboursId=....;

      r2=GetNorm(DisplacementVec);
      if(r2<LJCUTOFFsquared){
        Force=GetForces(DisplacementVec);
//...and modify only the forces acting on the thread's particle...
        MyForce.x=MyForce.x + Force*DisplacementVec.x;
        MyForce.y=MyForce.y + Force*DisplacementVec.y;
        MyForce.z=MyForce.z + Force*DisplacementVec.z;

        mypotential=mypotential+0.5*GetPotential(DisplacementVec);
      }
    }
    __syncthreads();
  }
//...which are saved in global memory at the end of the function call.
  Molecule[myparticleId].For.x=Molecule[myparticleId].For.x+Force*
      DisplacementVec.x;
  Molecule[myparticleId].For.y=Molecule[myparticleId].For.y+Force*
      DisplacementVec.y;
  Molecule[myparticleId].For.z=Molecule[myparticleId].For.z+Force*
      DisplacementVec.z;
  Molecule[myparticleId].Potential=mypotential;
}
```

Figure 4.5: Correct implementation of the force function.

Thread Branching

When a thread block is processed by the GPGPU, it is split into segments called warps. For each GPGPU model, there is a fixed warp size and each warp is sent to a GPGPU multiprocessor for simultaneous computation. The warp size of a 1.x compute capability device is 32. It is recommended to have a minimum of 64 threads in a block in order to avoid the idling of a multiprocessor during a function call. If one multiprocessor is finished with a warp before another, it can begin calculating the second warp assigned to it whilst the other multiprocessor catches up. In this manner multiprocessor de-synchroneity is cached.

When a warp is sent to a multiprocessor, it is verified whether every thread in the warp conducts the exact same operations. If it is found that a thread in the warp behaves differently due to asymmetric conditionalities, a thread branch occurs in which every thread in the warp needs to be evaluated sequentially. This means that an application with thread branching requires up to 32 times longer to compute a function as compared to an application without thread branching.

The code in Figure 4.5 clearly induces a large number of thread branches, since each thread in the cell has a unique position and therefore the distance from a proposed interaction partner is undeterministically below or above the cutoff radius.

Cell List Overhead

As the number of particles in a cell waxes and wanes, some mechanism must exist to ensure that all elements of a cell list are processed properly. Since an execution grid is launched with a constant, identical, number of threads per thread block, one is forced to use a fixed number of entries in each cell list and launch the execution grid in such a manner that each thread block consists of as many threads as there are entries in the cell list (in this case 64). Furthermore, it has to be ensured that at any given time, there are at least as many threads responsible for a given cell as there are particles inhabiting it. In order to quantify these considerations the following the following "overhead" is defined,

$$\text{Overhead} = \frac{\text{NumberofCellListEntries} \times \text{TotalNumberofCells}}{\text{TotalNumberofParticles}} - 1. \tag{4.1}$$

The cell size "overhead" is also equal to the relative difference between the number of threads launched by a block and the average number of particles in a cell. An overhead of 1 means that there are, on average, twice as many threads processing a cell as there are particles inhabiting it. When a block is launched, a number of excess threads will have no particle that they can be assigned to. These "ghost threads" are instructed not to calculate any interaction energies and are also not assigned a particle by means of an "if" condition. It must be noted, however, that these "ghost threads" serve a very important purpose because they participate in loading neighbor cell data. Each cell list has 64 entries and the entire list is loaded into the shared memory of the block if all threads partake in the loading sequence. This way, cells with variable populations still load the entire contents of a neighboring cell list.

Conclusion

A conventional CPU cell list algorithm produces the optimum performance if the simulation box is divided into as many cells as possible. This has the effect of reducing the average number of particles in a cell, which in turn means that each cell in the serial implementation of the algorithm searches smaller populations in order to find interaction partners for the particle that it examines at any given time. Although a maximization of the number of cells in a system has a similar effect on the CUDA implementation of the force function, one also requires the compilation and launch of many more threads and thread blocks each time the force function is called. In addition, a larger number of threads will take on the role of "ghost threads" resulting in more thread branches. In Chapter 5, these competing effects will be examined quantitatively to find whether a large number of cells with low populations and a large number of thread branches is favorable over a small number of cells with high particle populations and few thread branches.

The behavior of the Force functions is quite different from a serial implementation as a result of the aforementioned code and execution structure. It is notable that slight increases in the average cell density (corresponding to increases in system size) will have little effect on the calculation time. It is only after a critical cell density has been reached, resulting in fluctuations of cell populations that exceed the number of threads per block, that the number of cells in the system must be increased in order to accommodate for the additional particles.

4.1.3 The FillNeighbourLists Function

Although the FillNeighbourLists function has a virtually identical structure as the Force function, its purpose is quite different. Like its predecessor, this function evaluates the pair distance between a particle and its neighbors. Instead of calculating the interaction of the Lennard-Jones potential, its role is to analyze the geometry between the particles in accordance to the methods outlined in Chapter 2. The spherical functions employed for the bond order analysis require the cosine of the azimuthal angle as well as the polar angle of the displacement vector between two neighboring particles. These values are saved in a local array if two particles are sufficiently close to one another. After all calculations are completed, both aforementioned angles, the global id of each neighboring particle, and the number of neighboring particles are transfered to global memory slot dedicated to storing the local environment around a particle.

In order to extract the angles, it is necessary to employ the square root and the atan2 functions both of which are notoriously expensive and this is the reason why the FillNeighbourLists function requires more computational time than the Force function per call (see Figure 4.1). Since the interaction cutoff radius for the Lennard-Jones potential is larger than that of the neighbor cutoff (2.5σ for the former as opposed to 1.5σ for the latter), it might be sensible to employ an entirely different set of cell lists for this function. The advantage of a separate set of cells is that through the smaller neighbor cutoff radius, the cells would be smaller and

therefore feature fewer inhabitants. It is likely that the overhead of creating and maintaining a second set of cell lists would be overcome by the benefits of fewer superfluous calculations in this function.

4.2 Embarrassingly Parallel Functions

When a computational task can be parallelized highly efficiently, it is referred to as an "embarrassingly parallel" workload. This occurs, generally speaking, when the necessity for thread communication is miniscule and race conditions do not apply. Conversely, when the results of a thread's calculations are required by other threads for further calculations, a potential bottleneck arises in which all threads need to be synchronized (race conditions) and a data exchange is initiated at a limited transfer rate (thread communication). The more independent a thread is, the greater the rewards of parallelization. The following functions are deemed "embarrassingly" parallel because each thread has a well defined operation that it needs to conduct on the data irrespective of all other threads.

4.2.1 The Integration Function

Each particle in the simulation is represented by a struct, named "Molecule", the members of which contain all of the particle's properties. If an HMC move is applied, use of the three fields named "Pos", "Vel", and "For" which store a particle's position, velocity, and the forces acting on it, respectively (each of the aforementioned fields has an "x", "y", and "z" float member) is made. After calling the force function, which saves the forces acting on each particle in the "For" field, each "Molecule" has the Velocity Verlet algorithm applied to it. This is accomplished by launching an execution grid aimed at distributing all the particles in the system (*Number of Particles per Window* $\times$ *Number of Windows*) across the 16 multiprocessors as efficiently as possible. Each thread is instructed to load one particle's position, velocity, and the forces acting on it and apply the Velocity Verlet algorithm (see Chapter 3) on the data. Subsequently, periodic boundary conditions are applied.

An execution grid is employed consisting of 512 threads per thread block (the highest value allowed) and as many thread blocks as are necessary to have at least as many threads as there are particles. The largest system that was simulated consisted of 10976 particles per window and 20 windows. Therefore, 429 blocks were launched. The formula,

$$\mathrm{NumberofBlocksLaunched} = \frac{\mathrm{TotalParticles}}{512} + 1, \tag{4.2}$$

ensures that there are enough threads at all times. Any superfluous threads are instructed not to do anything. Thus the integration function is the proto-type of an embarrassingly parallel function. Its structure is depicted in Figure 4.6.

```
__global__ void Integrate(int VerletStep, Particle *Molecules,int
    BlockDim){

//Calculating the global thread Id for each thread, this corresponds to
     the particle that the thread will integrate.
  int threadid=threadIdx.x+BlockDim*blockIdx.x;

//A constant value that is loaded locally from global memory.
  double deltat=....;

//Depending on when the function is called, the argument VerletStep has
     either the value 1 or 2 which correspond to two the different
    steps, in the Velocity Verlet algorithm:

  if(VerletStep==1){
    Molecules[threadid].Pos.x += Molecules[threadid].Vel.x*deltat
    +0.5*Molecules[threadid].For.x*deltat*deltat;

    Molecules[threadid].Pos.y += Molecules[threadid].Vel.y*deltat
    +0.5*Molecules[threadid].For.y*deltat*deltat;

    Molecules[threadid].Pos.z += Molecules[threadid].Vel.z*deltat
    +0.5*Molecules[threadid].For.z*deltat*deltat;

    ....
//Imposing Periodic Boundary Conditions
    ....

    Molecules[threadid].Vel.x=Molecules[threadid].Vel.x
    +0.5*Molecules[threadid].For.x*deltat*deltat;

    Molecules[threadid].Vel.y=Molecules[threadid].Vel.y
    +0.5*Molecules[threadid].For.y*deltat*deltat;

    Molecules[threadid].Vel.z=Molecules[threadid].Vel.z
    +0.5*Molecules[threadid].For.z*deltat*deltat;
  }

  if(VerletStep==2){
    Molecules[threadid].Vel.x=Molecules[threadid].Vel.x
    +0.5*Molecules[threadid].For.x*deltat*deltat;

    Molecules[threadid].Vel.y=Molecules[threadid].Vel.y
    +0.5*Molecules[threadid].For.y*deltat*deltat;

    Molecules[threadid].Vel.z=Molecules[threadid].Vel.z
    +0.5*Molecules[threadid].For.z*deltat*deltat;
  }
}
```

Figure 4.6: The integration function presented schematically.

4.2.2 Other Embarrassingly Parallel Functions

The following functions are similarly simple to execute and program since each thread is responsible for evaluating a single particle's properties and all the necessary data can be retrieved from the appropriate part of the "Molecule" struct.

As was mentioned in subsection 4.1.3, the FillNeighbourlists function stores all the relevant data that is required to evaluate the q_{6m} bond order parameters (see Section 2.2) in the each "Molecule" struct. In the GPUCalculateQ6M function, each thread calculates the bond order parameter for a single particle. The same type of execution grid as with the integration function was launched in order to ensure that each particle is processed. Similarly, the EvaluateSolidity function computes the scalar product between each particle and its Frenkel neighbors in order to determine whether it is solid (again, see Section 2.2).

According to Chapter 3, the Hybrid Monte Carlo scheme requires a new velocity distribution to be drawn for the entire ensemble from an appropriate Gaussian distribution after each attempted HMC trial move. The open source GPGPU random number generator programmed by NVIDIA was modified to suit this implementation and exclusively makes use of the global memory, which is an advantage because the data is computed, stored, and retrieved exclusively via the GPGPGU global memory. Each thread in the function Sample Velocities draws from a cache of Gaussian random numbers and assigns them to the velocity components of the particle that it is responsible for. The forces and positions are saved in auxiliary arrays by the Backup function and if an HMC move is rejected then the RevertConfigurations function loads the stored positions and forces and reverts the array of "Molecule" structs to the former configuration. All of the aforementioned functions have the same execution grid and general structure of Integrate.

The Assign Cells function assigns to each particle in the system the cell number that it belongs to. Each window consists of n^3 cells and it is possible to calculate which of these cells a particle belongs to based on its x,y, and z coordinates. Each cell has a unique i.d. from 0 to $n^3 \times (\text{Numberofwindows} - 1)$ and the Assign Cells function saves into each particle's "Molecule" struct the cell i.d. that the particle belongs to.

The "global functions" such as NewKE and AcceptOrReject are launched by execution grids in which the thread blocks consist of one thread and the number of blocks launched is the number of windows in the simulation run. Quantities such as the acceptance probability of a volume or HMC move are evaluated or the kinetic energy of each window is concurrently computed.

4.3 Cell List Algorithms in CUDA

Once the Assign Cells function has finished, two routines are launched one after the other in order to update the cell lists. Both routines launch as many threads as there are cells in the simulation. The Leavers function searches all particles that are purportedly in a thread's cell and compares it with the index that the AssignCells

function has calculated, and if they are incongruent, the particles is deleted from the cellList. The Ids of the deleted particles are placed in an auxiliary struct for later use. The particles that have remained in their cells have their positions updated in the cell list that will be employed by the force function. Each thread in the Joiners function then searches the struct that contains all particles that have left their cells to append to its own list those particles that have entered it.

Chapter 5

Results

This chapter is dedicated to the quantitative analysis of the formalisms and algorithms that were introduced in the preceeding chapters. In Section 5.1, a comparison between the results obtained by the GPGPU implementation and those published by Mehlig *et. al.* [7] is made in order to verify whether single precision floating point operations are a hindrance in simulating the canonical ensemble correctly. In Section 5.2 the results on the nucleation of the Lennard-Jones liquid are presented and compared to those of Moroni's [6] and Frenkel *et. al.* [5] examinations. Finally, in Section 5.4 a comparison is made between the time the GPGPU took to arrive at the aforementioned results with the calculation time of a typical CPU. Throughout this chapter use of the reduced units, for which $\epsilon = \sigma = m = k_B = 1$, is made.

As an aid to the reader, the following distinctions are made. In Chapter 3, the HMC algorithm was introduced as an alternative to the RDMC method (see Section 3.3) and the act of propagating a configuration through phase space will be referred to as an *HMC step*. To simulate the NPT ensemble, the stochastic barostat introduced in Section 3.5 was implemented; when a simulation run conducts volume moves in combination with HMC steps, it is said to perform *NPT steps*. Finally, when the bond order analysis, the cluster analysis, and the biasing potential introduced in Chapter 2 are applied to the generated configurations after each NPT step, the simulation is said to undertake *nucleation iterations*.

5.1 Hybrid Monte Carlo Simulations

In this section, some of the results from Reference [7] will be recalculated and compared to the original work. The theoretical background of the HMC algorithm is treated in Chapter 3. The example set by Mehlig *et. al.* was followed by plotting the acceptance rate of the HMC algorithm as a function of the time step δt used to integrate the Hamiltonian equations of motion. In Figure 5.1, the acceptance rate of this implementation of the HMC algorithm applied to a Lennard-Jones system is compared to that of Mehlig's. Using 256 particles arranged in an fcc lattice, the system was equilibrated at a density of $\rho = 0.83$ and a temperature of $T = 0.72$. All simulations throughout this chapter made use of an interaction cutoff radius of

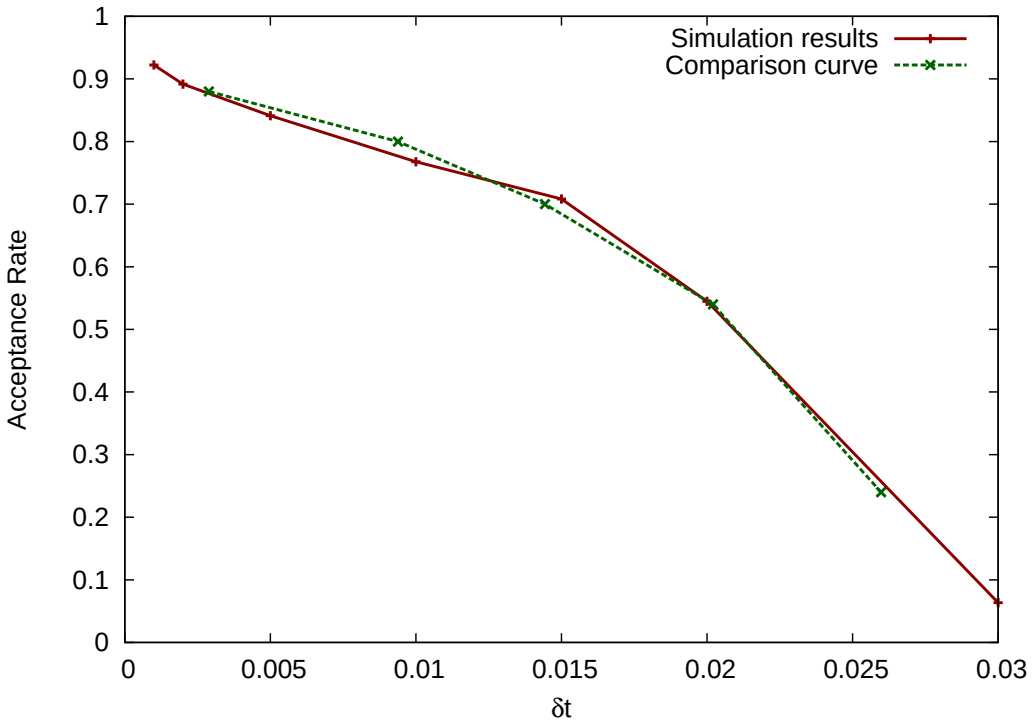

Figure 5.1: Comparison of HMC acceptance rates between our and Mehlig *et. al.*'s implementations for various discretizations of the equations of motion δt. The "comparison curve" (dotted line) was extracted from Reference [7] and the "simulation results" were obtained from the author's program.

$r_c = 2.5\sigma$. For this particular run, the error made through the use of a cutoff radius was not corrected for in either implementation. The algorithms sample the canonical ensemble. Following Mehlig *et. al.*, 10 Molecular Dynamics steps were conducted for all ensuing simulations before a trial configuration was subjected to the HMC acceptance criterion. The acceptance rate of an HMC move was calculated from a simulation run of 10000 HMC steps.

There is a point of minor confusion that arises from the use of different systems of units. As was mentioned, reduced units with $\epsilon = \sigma = m = k_B = 1$ were used by the author. Each of the aforementioned quantities refers to the properties of the Lennard-Jones particles and this specification defines the unit of time (most easily done over the definition of energy). Referring to their units as *scaled units* Mehlig *et. al.* cite the conventions used in Reference [24]. The paper in question is a study and extension of the phase diagram of argon through a series of simulations. It is supposed that the unit of energy was not set to ϵ but instead to 48ϵ so as to make the calculation of the Lennard-Jones forces easier. The relationship between the unit of time and the unit of length, energy and mass (denoted by t, L, E, and M respectively) is given by:

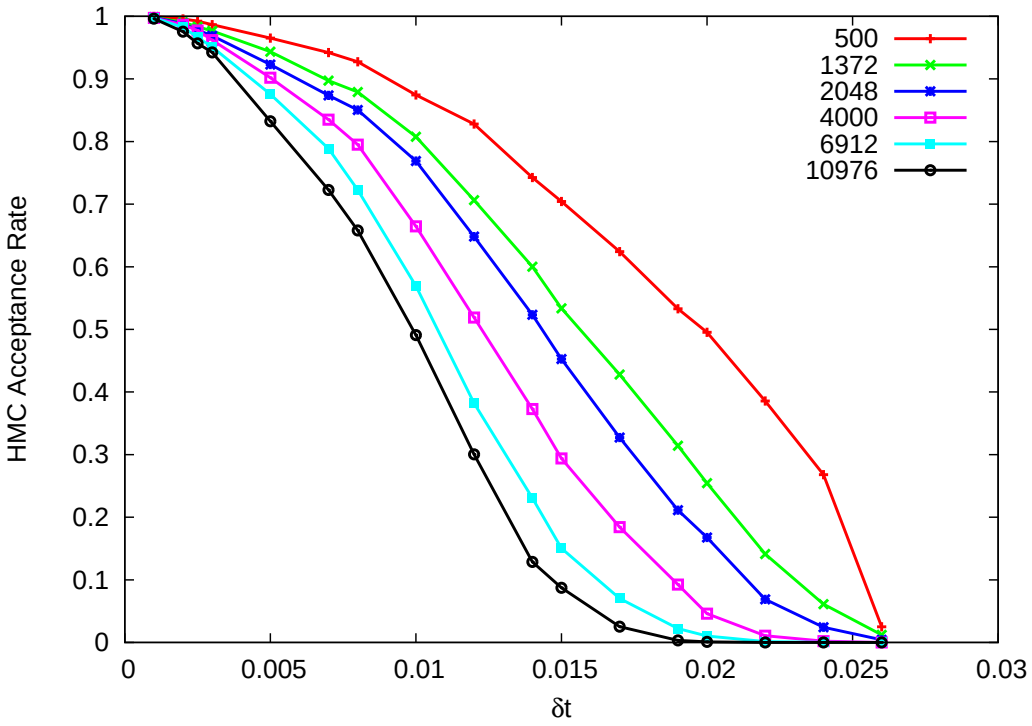

Figure 5.2: Acceptance probability of a trial configuration generated by the HMC algorithm as a function of the discretization δt. Each of the curves corresponds to a system of the indicated number of particles.

$$
\begin{aligned}
[E] &= [M] \times \frac{[L^2]}{[t^2]} && (5.1)\\
&\textit{(dropping the square brackets)} \\
t^2 &= L^2 \times \frac{M}{E} \\
t &= L \times \sqrt{\frac{M}{E}}. && (5.2)
\end{aligned}
$$

Thus the unit of time scales with the square root of the unit of energy, which in this case, is 48 times smaller than Mehlig's. By dividing the time axis in Mehlig's paper by the square root of 48, a conversion between the two systems of units is achieved and the appropriate comparison in Figure 5.1 can be made.

While testing HMC algorithm, it was found that the HMC acceptance rate depends on the size of the simulated system. In Figure 5.2, the acceptance rates of the HMC algorithm are plotted as a function of the discretization step for various system sizes. The initial configuration of the system was equilibrated for 1000 HMC steps with $\delta t = 0.005$. The acceptance rate was tracked for 4000 HMC steps using various values of δt ranging from 0.001 to 0.026. The simulations were conducted at $T = 0.72$ and a density $\rho = 0.83$.

Figure 5.2 shows that larger systems suffer from lower acceptance probabilities for a given δt. The Velocity Verlet algorithm incurs a global error of order δt^2 in calculating the new position of a particle each iteration and a global error of order δt^2 in the calculation of a particle's velocity. This error propagates non-linearely in the evaluation of the potential and kinetic energy and results in the change of the system's Hamiltonian, $\Delta\mathcal{H}$ (which tends zero in the limit $\delta t \to 0$). It is therefore to be expected that the numerical error in the evaluation of the Hamiltonian of the system grows with the number of particles since each particle contributes to $\Delta\mathcal{H}$. The errors associated with the Velocity Verlet algorithm are not additive in the calculation of kinetic and potential energies, resulting in the algorithm's stability even for a large number of MD steps. An additional effect of the non-linearity of error propagation is that the acceptance probability does not decay linearly with respect to the system size. Another, perhaps more relevant, consequence of increasing the system size is that the thermal fluctuations also grow, increasing the values of $\Delta\mathcal{H}$ and therefore reducing the acceptance rates.

In order to reproduce the simulations of Frenkel *et. al.* and Moroni [5,6], it was necessary to implement a barostat (see Section 3.5). To verify the accuracy of the implementation, a recalculation of two points of the Lennard-Jones phase diagram cited by Frenkel *et. al.* [5] was attempted. The original simulations were conducted by Hansen and Verlet [25]. In order to retrieve the correct density at the specified temperature and pressure, it was necessary to employ the potential shift and the tail corrections described by Frenkel and Smit [17]. In addition to reproducing the phase diagram of the Lennard-Jones system faithfully, the tail corrections also had a dramatic effect on the acceptance rate of HMC trial moves. When employing a potential shift, the acceptance rate of the HMC algorithm increases considerably and the dependence on the system size is suppressed. The systems in Figure 5.2 all benefited from tail corrections.

Although Mehlig *et. al.* found that an acceptance rate of 70% induces the optimal decorrelation rate of the sampled configurations, the simulations intended to reproduce the free energy of nucleation (consisting 10976 particles) were carried out for a δt of 0.008, corresponding to an acceptance rate around 60%, because it was found that the windows sampled their respective intervals most efficiently in that regime.

Another phenomenological behavior was discovered during the process of equilibration. All simulation runs were initiated by an assortment of particles arranged in a perfect fcc lattice that was equilibrated before any measurements were taken. It was found that the acceptance rate during this equilibration phase drops dramatically and therefore the algorithm takes a comparatively long time to bring the system to equilibrium. As a result, it was deemed more sensible to use a small δt for the equilibration run after which the ensuing configurations were used as the starting point for a production run with a larger δt.

In summary, a discussion of the dependence of the HMC acceptance rate on the system size (Figure 5.2) and the observation that tail corrections to the Lennard-Jones potential raise the acceptance ratios of HMC trial moves is added to the

analysis conducted by Mehlig *et. al.*.

5.2 Nucleation Results

5.2.1 NPT-Simulations

In testing the accuracy of the NPT algorithm, simulation runs were initiated for which a volume move was attempted with a probability 30%, a relatively large rate compared 5% volume move rate that sufficed for the production runs that generated the free energy curve of nucleation (see Section 5.3). The test simulations consisted of 10976 particles, with $\delta t = 0.003$, and the ensemble was equilibrated for 1000 NPT steps before the resulting configuration was used as a starting point for a 6000 NPT step simulation from which the average densities were calculated. In Figure 5.3, the density fluctuations of the Lennard-Jones simulations are presented for two pairs of pressure and temperature on the coexistence curve. For each of the two points on the coexistence curve that were examined, an equilibrated fcc lattice and an equilibrated liquid was prepared and sampled. It was found that the average values of these densities exhibit a good correspondence to the literature values from Reference [25], a summary of which can be found in Table 5.1.

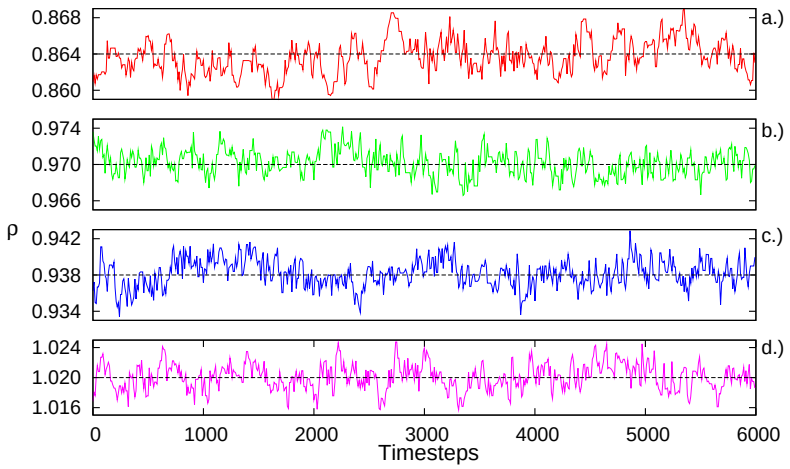

Figure 5.3: Density fluctuations as a function of time for 10976 Lennard-Jones particles. The system was prepared in the: a.) liquid phase at $T = 0.75$ and $P = 0.67$ b.) fcc phase at $T = 0.75$ and $P = 0.67$ c.) liquid phase at $T = 1.15$ and $P = 5.68$ d.) fcc phase at $T = 1.15$ and $P = 5.68$. The average density was evaluated, plotted, and labelled in each graph as the dashed black line.

Pressure	Temperature	Phase	ρ^*	ρ
0.67	0.75	liquid	0.875	0.864
		solid	0.973	0.970
5.68	1.15	liquid	0.936	0.938
		solid	1.024	1.020

Table 5.1: Comparison of the densities calculated by Hansen and Verlet [25] denoted as ρ^* and the densities calculated by the HMC algorithm denoted as ρ obtained from the averages of the simulations in Figure 5.3. The largest deviation from the literature values can be found in the first entry of the table and corresponds to a relative error of 1.2%.

5.2.2 Bond Order Parameters

In this section, the recalculation of the distributions published by Frenkel *et. al* and Moroni [5, 6] is presented and discussed. In what follows, the behavior of the particles' q_6 bond order parameter, the dot products between their $\vec{q}_6$ vectors, and the number of solid bonds is examined from simulations conducted on the coexistence curve and compared to the results obtained by the aforementioned authors.

The systems employed consisted of 4000 particles that were integrated with a time step δt of 0.005. A volume move was attempted with a frequency of 10% and all systems were equilibrated for 2000 steps. Every 200 NPT steps, the value of each particle's q_6 bond order parameter, one of the dot products between the nearest neighbors of each particle, and the number of solid bonds was recorded. The data for each run was collected over a simulation run of 9000 NPT steps. In order to reproduce the systems that were examined by Moroni and Frenkel, the ensemble had to be simulated using various temperatures, pressures, and the neighbor cutoffs.

When Frenkel *et. al.* [5] tested their bond order analysis algorithms, the distributions of the q_6 order parameter were obtained at a pressure $P = 5.68$ and a temperature of $T = 0.92$. Although a neighbor cutoff of 1.5σ is reported, comparisons with simulation results indicate that the cutoff they used was most certainly 1.4σ (this observation is corroborated by Moroni [6]). In Figure 5.4, the recalculated q_6 distribution (left) is compared to the distribution of Frenkel *et. al.* (right). The correspondence of the shape of the curves, the values of their maxima, their points of intersection, and the characteristic kinks are pleasingly consistent. This is the first step in distinguishing the solid Lennard-Jones phase from the liquid phase.

In Figure 5.5, a comparison of the distribution of the dot products of each particle with its nearest neighbors for three phases at coexistence using $T = 1.15$, $P = 5.68$ and a neighbor cutoff of 1.5σ is made. The comparison curve was obtained from the work done by Moroni [6]. As the distribution of the dot products suggests, a particle bond with a dot product greater than 0.5 will almost certainly belong to two particles that are in the solid phase. The reader will note that the similarity of the graphs extends to their heights and points of intersection. The roughness of the curve for the liquid and bcc phase is attributed to the difference in the histogram bin width since it is unlikely that its source is statistical.

Since both authors simulated at $T = 1.15$ and $P = 5.68$ in evaluating the distribution of solid bonds, one is able to examine the effect of the neighbor cutoff radius because that is the only difference between the two systems. In Figure 5.6, a neighbor cutoff of 1.4σ was employed, which results in a sharp maximum at 12 for the fcc curve and a much smaller one at 13 for the bcc phase. In Figure 5.6, on the other hand, the maximum of the fcc curve is at 13 and the bcc curve has a more pronounced peak. The difference is induced solely through the use of a neighbor cutoff of 1.5σ. The disparities between the shapes of the curves in Figures 5.6 and 5.7 are irrelevant for simulation purposes because, in either case, particles with more than 8 solid bonds are virtually always the solid phase.

Since each particle in the simulation contributes towards the calculation of the

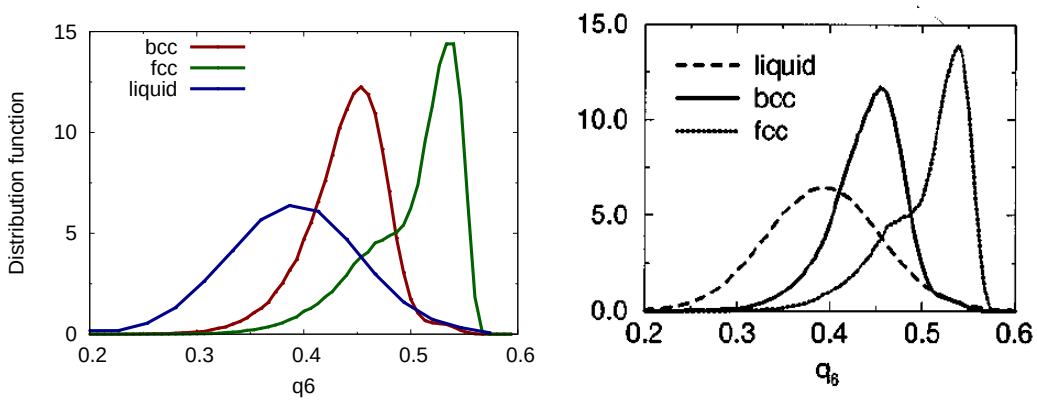

Figure 5.4: Left: distribution of q_6 bond order parameter obtained through simulation. Right: Original q_6 distribution first published in [5].

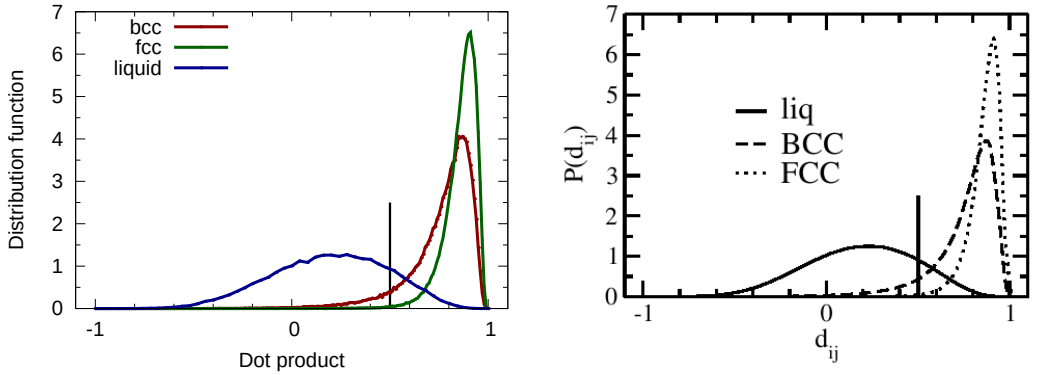

Figure 5.5: Left: recalculation of the distribution of the dot products between nearest neighbors. Right: Curve obtained from [6].

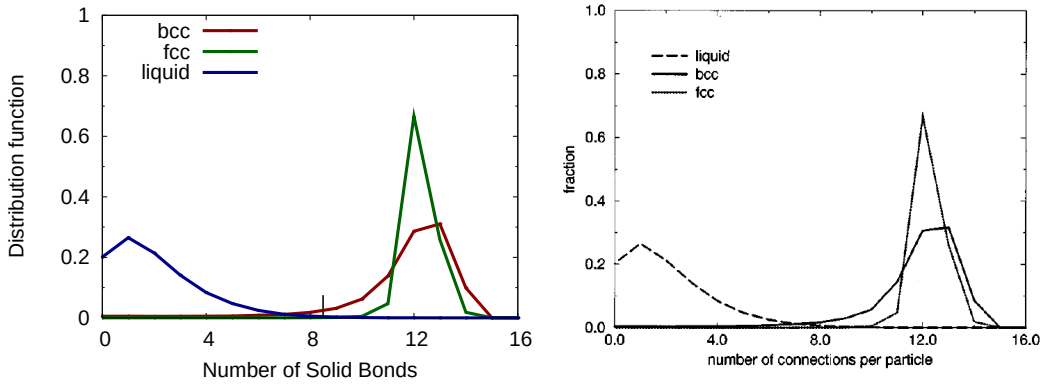

Figure 5.6: Left: Distribution of each particle's solid bonds for a neighbor cutoff of 1.4σ. Right: Curve obtained from [5].

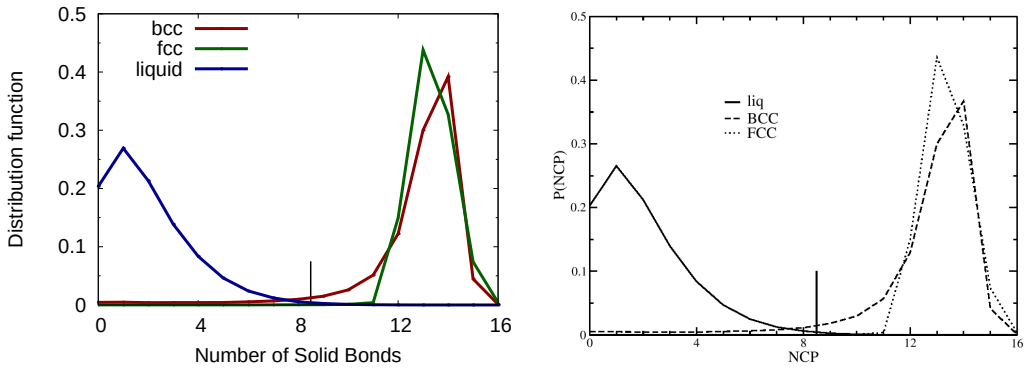

Figure 5.7: Left: Distribution of the number of each particle's solid bonds for a neighbor cutoff of 1.5σ. Right: Curve obtained from [6].

distribution, one is able to achieve accurate averages after comparatively few iterations. This is why it is somewhat unsurprising that the correspondence between the simulated and comparison curves is so good. In addition to showing that the analysis of the number of solid bonds is an effective criterion in distinguishing between the solid and liquid phase, a satisfying correspondence with the accepted literature values has been established.

5.3 The Free Energy of Nucleation

Equilibration

Although the nucleation iteration algorithm has been developed to keep a series of NPT simulations within a given segment of the reaction coordinate, one still needs a procedure to bring Lennard-Jones ensembles within the correct range in the first place. A simple proceedure was developed: for each window, one begins with an fcc crystal and chooses an arbitrary particle in the lattice about which a spherical collection of particles is constructed, the size of which corresponds to the bias center of the window that it belongs to. With exception of the aforementioned particles, the entire system is then integrated at a high temperature in order to melt the remainder of the lattice. After the system is equilibrated the ensuing configurations are used as the starting point of a second run in which all particles are free to move and the temperature is set to the value for which one intends to calculate the free energy barrier. Each window thereby equilibrates its configuration while maintaining an appropriate cluster size.

The Free Energy Barrier

Although both Frenkel *et. al.* and Moroni proceed to calculate the free energy barrier of the nucleation of a Lennard-Jones liquid, the former evaluated the free energy as a function of the average crystallinity of the system (the average q_6 per particle) whereas the latter employed a cluster analysis and calculated the free energy as a

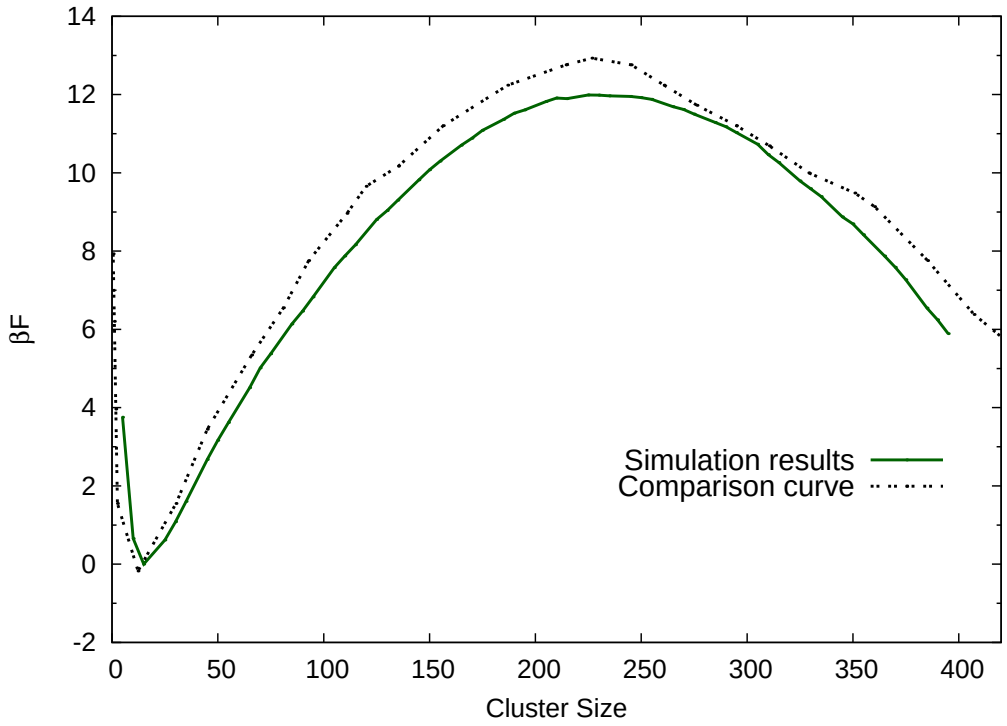

Figure 5.8: The free energy of nucleation of a Lennard-Jones liquid at $T = 0.89$ and $P = 5.68$. The comparison curve has been obtained from [6].

function of the largest cluster in the system. The advantage of using the average crystallinity is that it is a global property of the system and does not suffer from the simulation artifact that is discussed in 2.3.1. This advantage is, in a way, also a drawback since the details of the structure of the solid nucleus is hidden by averaging over the entire system. In addition, the position of the ensuing free energy curve is likely to change with the system size because an increase in the system size will shift the average q_6 towards the liquid phase without changing the likelihood of obtaining larger clusters in the same measure. For these reasons, the procedures detailed by Moroni [6] shall be followed.

To calculate the free energy of nucleation at a temperature of $T = 0.83$ (corresponding to 25% undercooling) and a pressure of $P = 5.68$, 20 simulations of 10976 particles each, with differing bias centers in every window, were carried out concurrently. The neighbor cutoff was set to $r_{nc} = 1.5\sigma$ and volume moves were attempted with a frequency of 5%. The HMC time discretization was set to $\delta t = 0.008$. Bias centers in steps of 20 particles were assigned to each window, beginning at a cluster size of 10 and ending at 390. An fcc lattice was melted for 3500 NPT timesteps using the equilibration process described in the previous section and the system was simulated for $10,000$ nucleation iterations. Once the equilibrated configurations were produced, a simulation run of $250,000$ nucleation iterations was initiated. After every 10 nucleation iterations the size of the largest clusters of each window were recorded and histogrammed.

After unbiasing the probability distributions that were produced by the simulation, the "self consistant histogram method" [17] was used in order to optimize the overlap between the free energy segments produced by each window. The full free energy curve is presented in Figure 5.8 next to the free energy curve calculated by Moroni [6]. It is difficult to infer the reason for the discrepancy between the two curves but it is supposed that the difference in the system sizes might account for the shift. The reference curve was obtained from simulations of 10648 particles whereas the simulated curve was produced by systems of 10976 particles. Since the latter system is about 3% larger than the former, it might be possible that the probability of finding larger clusters is increased somewhat thereby reducing the height of the free energy barrier. This supposition is supported by the fact that shifting the simulated curve by a cluster value of 3 makes the positions of the "free energy wells" coincide perfectly and, coupled with a tranlslation of the simulated curve by 0.5β, the overlap between the two curves is maximized. In both cases, the maximum of the free energy barrier lies at a cluster size of about 240 particles which is why, in combination with the correspondance of the previous graphs, the results are considered trustworthy. The difference in the height of the two barriers is estimated to be around 8%.

By simulating a single window for 250000 nucleation iterations and tracking all of the solid clusters that appear in the accepted configurations, one can remove the simulation artifact that appears in at the beggining of the graph in Figure 5.8.

The difference in the calculation of the free energy barrier is far more pronounced in Figure 5.9 than in Figure 5.8; however shifting the simulated curve by 6β produces a profound overlap between the two curves (see Figure 5.10). Since the deviation of the two free energy curves seems to systematic over the entire interval of the reaction coordinate, it is supposed that the simulated system, which is larger than the comparison system, generates crystal nuclei with a greater probability.

5.4 Speedup Factors

The first dependence that shall be examined is the relationship between the total computational time for a simulation run as a function of the number of windows that were simulated. It is suggestive that there is a linear relationship between the two, which makes it possible to conduct subsequent measurements using a single simulation box and extrapolating the data for multiple windows. The systems consisted of 4000 Lennard-Jones particles arranged in an fcc lattice that was equilibrated at a temperature $T = 0.83$ and a pressure $P = 5.68$. The program was executed for 10000 nucleation iterations (with $k_n = 0$) and the computational time was recorded for umbrella sampling simulations ranging from 1 to 20 windows. In Figure 5.11 all the calculation times are divided by the time taken to simulate a single window for 10000 nucleation iterations and the ensuing factors appear on the y-axis. On the x-axis the number of windows in the simulation run are labelled. A linear fit was applied to the data.

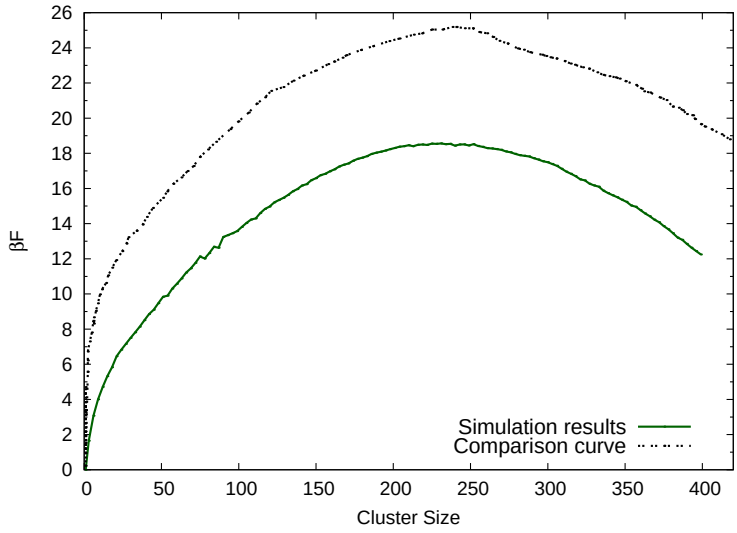

Figure 5.9: The free energy of nucleation of a Lennard-Jones liquid at $T = 0.89$ and $P = 5.68$ in which the first window was simulated without a biasing potential and all clusters were taken into account. The free energy profiles of the remaining windows were then "pasted" to that of the initial window. The comparison curve has been obtained from [6].

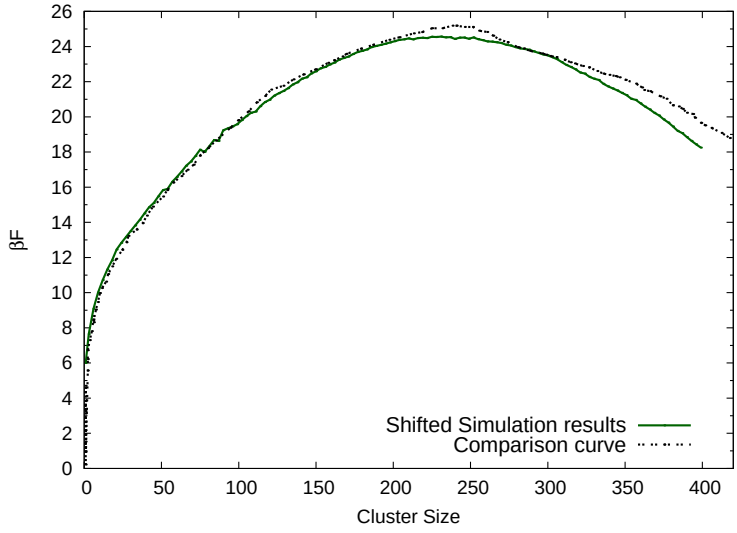

Figure 5.10: The free energy of nucleation of a Lennard-Jones liquid at $T = 0.89$ and $P = 5.68$. Using the same proceedure as in Figure 5.9, the simulated curve was shifted by 6β in order to show that its shape corresponds very well to the comparison curve.

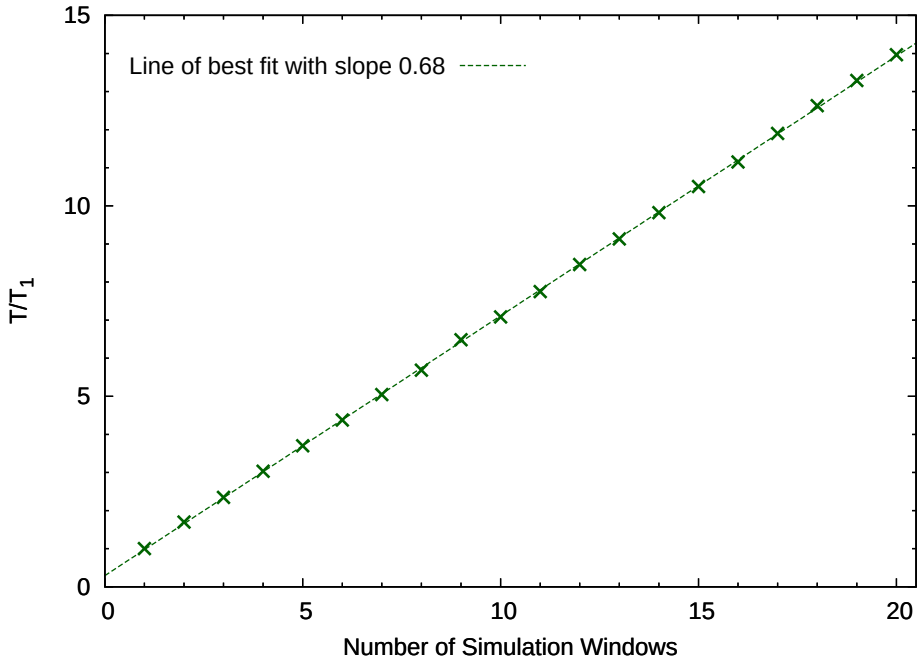

Figure 5.11: Runtime vs number of simulation windows. The calculation times are divided by the time taken to simulate one window for 10000 nucleation iterations $T_1 = 378.751$ seconds. The shape of the graph clearly indicates a linear relationship between the number of simulated windows and the runtime. A line of best fit with a slope of 0.68 and a y-intercept of 0.29 has been plotted through the data points.

The 10000 nucleation iterations take far longer to execute than the initialization sequence the influence thereof is therefore negligible. Although there is a clearly linear relation between the calculation time and the number of simulated windows the slope of 0.68 suggests that the marginal cost of compiling and launching additional threads and blocks decreases with the addition of windows. It is posited that this deviation of the slope from unity is responsible for the nonzero value of the y-intercept. The results obtained from the data in Figure 5.11 will be useful in comparing the GPGPU performance with that of a cluster of CPUs.

Of all the parameters associated with the simulation, the most influential in the calculation time was the number of cells that the simulation box was decomposed into. The allusion from Section 4.1.2 regarding the loss of computational power as a result of thread branching will be quantitatively examined. To this end, the term "basecells" will be used henceforth to refer to the cubic root of the number of cells that a window is divided into. A "basecell" value of 3 implies that a window is decomposed into 27 cells. Using the volume of a simulation box and its particle number one can calculate both the largest and smallest number of basecells allowed. As the simulation is divided into more cells, the length of the sides of a cell shrinks until it is shorter than the cutoff radius and this sets an upper bound to the number of basecells that can be employed. At the same time, enough basecells must exist so

that the average number of particles does not exceed 64 (this value shall be justified shortly) since this value corresponds to the number of cell list entries that were used in the program.

In the analysis that follows, it will be shown that the GPGPU is generally most efficient when the number of basecells is minimized. The significant pitfall of configuring a system with a low "overhead" (see Section 4.1.2) is that although, on average, fewer than 64 particles inhabit a cell, the local density fluctuations may result in a cell becoming overfilled thereby inducing a segmentation fault. In the author's experience, an overhead as low as 10% has been sufficient to avoid a system crash even for long simulation runs and therefore configurations with overheads of 20% or more are thoroughly stable. It ought to be reiterated that the contents of this paragraph would be irrelevant if it were not for that fact that the application performs somewhat better if the number of basecells is minimized.

Using these two criteria as the upper and lower bound of the number of cells a system may adopt, simulation runs of multiple system sizes were initiated in order to document the effect of the number of basecells on the calculation time. The simulations were equilibrated 1000 steps with $\delta t = 0.001$ and then a run of 10000 nucleation iterations was initiated using $\delta t = 0.005$. The temperature was set to $T = 0.83$ and the pressure was $P = 5.68$. Additional test runs were conducted at higher temperatures for which the liquid phase formed and the results were essentially unchanged. This examination's results are presented in Figure 5.12. With singular exceptions in the curves for the systems of 2916 and 16384 particles, all calculation times increase with rising number of basecells. Thus the thread branching effect discussed in Section 4.1.2 takes precedence over the superfluous calculations incurred by making cells "unnecessarily" large. In addition, the reduction in the number of blocks launched to calculate the forces also benefits the computational time. The following rule of thumb is posited: "The lower the overhead, the fewer thread branches occur which in turn optimizes the calculation time". As a result, 64 threads per block seem optimal since, on the one hand, the thread branches are kept at a minimum and yet multiprocessor desynchroneity is also suppressed (see Section 4.1.2).

It is somewhat irksome to discover that the cutoff radius and the density (by affecting the number of particles in a cell) of the system play a role in determining the optimal parameters and the calculation time of the simulation. The decrease in the performance is of the order of 10% for systems below 10976 particles and as high as 27% for the system of 16384 particles.

Using the optimal number of basecells, simulation runs of different system sizes were initiated using the same parameters as with the basecell analysis and compared to the performance of a CPU. Contrary to the GPGPU implementation, the CPU comparison application made use of the maximum number of basecells available in order to optimize the run time. The code that was used as a comparison was as similar as possible to the code executed by the GPGPU. Any deviations from the original were implemented to make the CPU code more competitive. The CPU code was executed on a single processor of an Intel^{R} Core^{TM} Duo 6600 (at 2.4 GHz) in

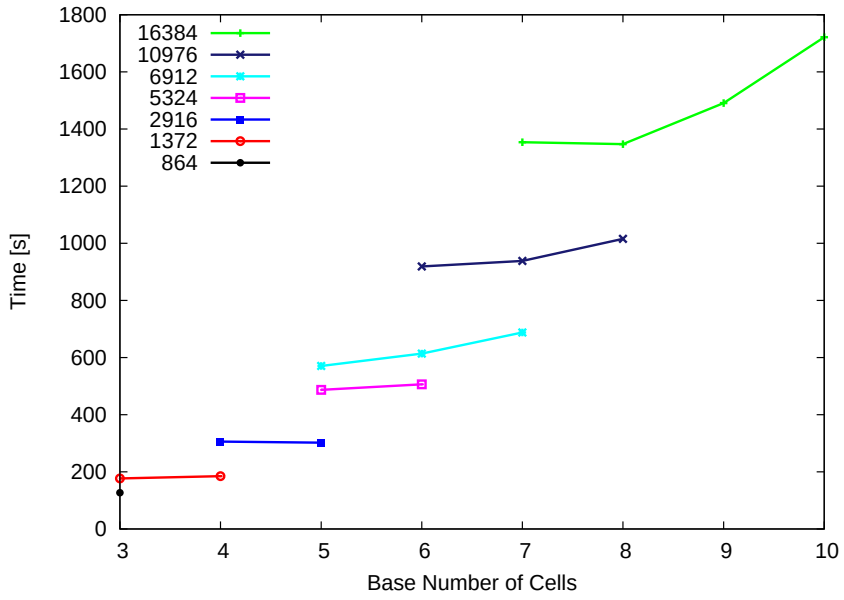

Figure 5.12: Calculation time for 10000 nucleation iterations as a function of the number of basecells for multiple system sizes.

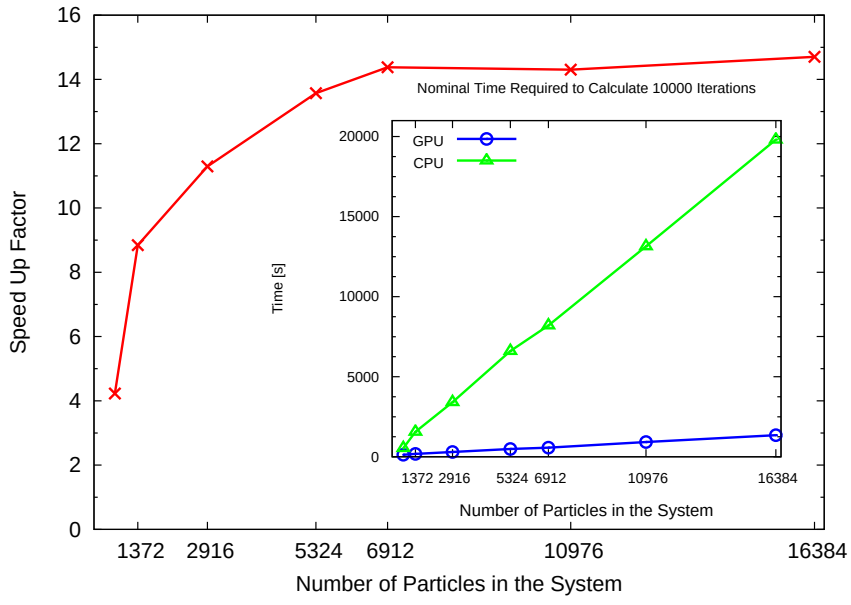

Figure 5.13: Speedup factor as a function of system size. The speedup factor is defined as the quotient of the time taken by the CPU and the GPU to calculate 10000 iterations of the algorithm. In the inset, each machine's nominal calculation time is plotted as a function of system size.

order to simulate the action of a single node in a cluster of such machines. The speedup that is plotted as a function of system size in Figure 5.13 is calculated using,

$$\text{Speedup} := \frac{\text{T}_{\text{CPU}}}{\text{T}_{\text{GPU}}}. \tag{5.3}$$

5.5 Summary

A form of "trivial parallelization" can be undertaken by assigning a single window to a node on a conventional CPU cluster as was done by Lechner [4] using the Message Passing Interface (MPI) for C. Since 20 windows were simulated in this umbrella sampling simulation and the calculation time of the GPU scales linearly with the number of windows in the system (see Figure 5.11), a speed up factor of 20 would imply that the GPGPU completes the calculation of the free energy barrier as quickly as a CPU cluster. Since the speedup factor for a system of 10976 particles is close to 14 (Figure 5.13), a single Tesla C870 GPGPU (which is, in the meantime, somewhat outdated) is able to achieve 70% of the computational efficiency of a 20 node cluster of Intel$^{\text{R}}$ Core$^{\text{TM}}$ Duo 6600 processors.

Chapter 6

Final Remarks and Outlook

The question as to the viability of CUDA enabled devices as an alternative to conventional CPUs can be reformulated in three parts. The first is whether the limitation of float precision would be an obstacle towards correct results. The second doubt is whether memory constraints and data communication are so prohibitive as to make an implementation unfeasible. Finally, the question arises whether the speedup, if at all measurable, is worth the effort of learning CUDA and using it to implementing such a simulation experiment.

It is difficult to make generalizations about methods that apply to the physics, numerics, and computational science at the same time. However, for the case in point, the results are pleasing. In this instance, the free energy curve was calculated to a high degree of accuracy, with the results of the conducted simulations corresponding well with the host of comparison curves that were available. The effort of learning CUDA was minimal since the author has had occasion to learn and program in C, so the crossover required a minimum of study and yet the computational time was reduced by an order of magnitude through the use of a single GPGPU, producing a comparable performance to that of a cluster CPUs. Finally, with exception of one particular quantity (the total potential of the system for which a trivial change was necessary), no adjustments had to be made to account for the lack of precision offered by floats. It is therefore proposed that this proof of principle has been rather successful in confirming that CUDA GPGPUs are a viable option for the simulation of condensed matter systems.

The investment in larger memory capacities, the improvement of internal and external GPGPU data communication, and the introduction of double precision calculations (all of which are natural and necessary for the progression of graphics processing) ought to ensure that GPGPUs have a promising future ahead of them for computational physics as well.

Bibliography

[1] J.A van Meel. MDGPU. *http://www-old.amolf.nl/~vanmeel/mdgpu/about.html*, 2008.

[2] Joshua A. Anderson. Highly Optimized Object-Oriented Molecular Dynamics. *http://www.external.ameslab.gov/hoomd/index.html*, 2008.

[3] J. A. van Meel and A. Arnold and D. Frenkel and S. F. Portegies Zwart and R. G. Belleman. Harvesting graphics power for MD simulations. `arXiv/0709.3225` *[cond-mat.soft]*, 2007.

[4] Wolfgang Lechner. *Nucleation and Defect Interactions in Colloidal Suspensions.* Universität Wien, 2009.

[5] P.R. ten Wolde, M.J. Ruiz-Montero, and D. Frenkel. Numerical calculation of the rate of crystal nucleation in a lennard-jones system at moderate undercooling. *J. Chem. Phys.*, 1996.

[6] Daniele Moroni. *Efficient Sampling of Rare Event Pathways, PhD thesis.* Universiteit van Amsterdam, 2005.

[7] B. Mehlig, D. W. Heermann, and B. M. Forrest. Hybrid monte carlo method for condensed-matter systems. *Phys. Rev. B*, 45(2):679–685, Jan 1992.

[8] NVIDIA. NVIDIA CUDA Zone. *http://www.nvidia.com/object/cuda_home.html*, 2008.

[9] Vijay Pande. Folding at home faq. *http://folding.stanford.edu/English/FAQ-highperformance*, 2010.

[10] NVIDIA. Cuda downloads. *http://www.nvidia.com/object/cuda_get.html*, 2009.

[11] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Phys. Rev.*, 65(3-4):117–149, Feb 1944.

[12] A. Gambassi S.Dietrich & C. Bechinger C.Hertlein, L.Helden. Direct mesurement of critical casimir forces. *Nature*, 451.

[13] David Chandler. *Introduction To Modern Statistical Mechanics.* Oxford University Press, Berkeley, 1987.

[14] J. S. van Duijneveldt and D. Frenkel. Computer simulation study of free energy barriers in crystal nucleation. *The Journal of Chemical Physics*, 96(6):4655–4668, 1992.

[15] Paul J. Steinhardt, David R. Nelson, and Marco Ronchetti. Bond-orientational order in liquids and glasses. *Phys. Rev. B*, 28(2):784–805, Jul 1983.

[16] William Humphrey, Andrew Dalke, and Klaus Schulten. VMD – Visual Molecular Dynamics. *Journal of Molecular Graphics*, 14:33–38, 1996.

[17] Daan Frenkel and Berend Smit. *Understanding Molecular Simulation*. Academic Press, New York, 2002.

[18] M. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. N. Teller, and E. Teller. Equation of state calculations by fast computing machines. *J. Chem. Phys.*, 21:1087, 1953.

[19] David Chandler. *Introduction to Modern Statistical Mechanics*. Oxford University Press, 1. edition, 1987.

[20] Simon Duane, A. D. Kennedy, Brian J. Pendleton, and Duncan Roweth. Hybrid monte carlo. *Physics Letters B*, 195(2):216 – 222, 1987.

[21] Alan M. Ferrenberg and Robert H. Swendsen. Optimized monte carlo data analysis. *Phys. Rev. Lett.*, 63(12):1195–1198, Sep 1989.

[22] Alan M. Ferrenberg and Robert H. Swendsen. New monte carlo technique for studying phase transitions. *Phys. Rev. Lett.*, 61(23):2635–2638, Dec 1988.

[23] NVIDIA. Nvidia cuda programming guide. *http://developer.download.nvidia.com/compute/cuda/2_3/toolkit/docs/*, 2009.

[24] JJ Nicolas, KE Gubbins, WB Streett, and DJ Tildesley. Equation of State for the Lennard-Jones Fluid. *Molecular Physics*, 37(5):1429–1454, 1979.

[25] Jean-Pierre Hansen and Loup Verlet. Phase transitions of the lennard-jones system. *Phys. Rev.*, 184(1):151–161, Aug 1969.

Appendix

A.1 Zusammenfassung

Diese Masterarbeit untersucht die Anwendbarkeit von CUDA Graphikkarten in modernen Physik Simulationen. Zwei bestehende CUDA Projekte, die man auf der NVIDIA Website finden kann, und eine Publikation von J.A. van Meel dienen als die Inspiration dieses Unterfangens.

Das untersuchten Algorithmen sind der Ausgangspunkt der detaillierten Analyse der Nukleationsmechanismen von unterkühlten Lennard-Jones Flüssigkeiten. Die Voraussagen der klassischen Nukleationstheorie werden durch die Messung der freien Energie als Funktion der Größe der kristallinen Kerne, die sich in der unterkühlten Flüssigkeit bilden, bestätigt. Die Arbeit von Wolfgang Lechner in seiner Analyse des "Gaussian Core" Modells dient als Anhaltspunkt dieser Arbeit, wobei Frenkel et. al. und Daniele Moroni als die ursprünglichen Architekten dieser Methoden gelten. Um die Nukleation von unterkühlten Flüssigkeiten zu untersuchen, wurden, zusätzlich zu einem Monte Carlo Simulationsalgorithmus, "Bond Order Analysis" und "Umbrella Sampling" Algorithmen in CUDA entwickelt. Die Komplexität des Systems und die zu seiner Untersuchung notwendigen Algorithmen sind ein ausreichend guter Test, um die Brauchbarkeit von CUDA GPGPUs in modernen Physik Simulationen zu evaluieren.

Unter Verwendung einer NVIDIA Tesla C870 GPGPU konnten trotz der EinschrÃd'nkungen durch die "floating point" Genauigkeit der Graphikkarte mit der Literatur übereinstimmende Resultate produziert werden. Diese Resultate wurden um ungefähr einen Faktor 10 schneller errechnet als auf einem Intel Core Duo 6600 Rechner. In diesem Fall ist damit die Leistung einer solchen Graphikkarte vergleichbar mit der Leistung eines Clusters von Intel Core Duo 6600 Rechnern.

Curriculum Vitae

Personal Data:

Given Name:	Syed Jaffar Mehdi Hasnain
Date of Birth:	25/09/1983
Citizenship:	Canada
Place of Birth:	Vienna, Austria

Academic History:

06/2002:	International Baccalaureate (IB) graduation with 39/42 points
08/2002-06/2003:	Enrolled at McGill University physics honors program
10/2003-11/2008:	Enrolled at University of Vienna physics program for the Magista degree
03/2009-12/2009:	Enrolled and completed University of Vienna physics bachelor's program
15/12/09:	Enrolled in University of Vienna physics master's program
29/01/09:	Application for master's orals to obtain an Msc in physics

Extra Educational Activities:

2003-2009:	Tutoring for Physics and Mathematics for the Austrian Matura exam (high school)
2008:	Junior researcher in the simulation group at the Austrian Competence Center for Tribology (AC^2T)
10/2007-10/2008:	Tutor for the department of physics at the University of Vienna for the lecture “Introduction to Mathematics for Physicists”

Spoken Languages:

English (fluent), German (fluent), French (spoken and read), Bahasa Indonesia (spoken and read), Urdu (comprehended)

Title of the Master's Thesis: “Enhanced Window Sampling Using CUDA Enabled Devices”

Residential History:

1983-1996:	Vienna, Austria
1996-2002:	Jakarta, Indonesia
2002-2003:	Montreal, Canada
2003-present:	Vienna, Austria