Parallel Implementation of Tail Correction in Energy-Pressure for Molecular Simulations

Stefanos Karandreas

Bachelor Thesis

Department of Physics - University of Crete
Heraklion, Crete 2016

Thesis Advisor: Dr. Vagelis Harmandaris

Thesis Committee

Dr. V. Harmandaris - Department of Mathematics and Applied Mathematics
Dr. I. Kominis - Department of Physics
Dr. N. Christakis - Department of Physics

Acknowledgements

I would like to thank Dr. Vagelis Harmandaris for providing me with the opportunity to produce this thesis and sharing his vast knowledge on molecular dynamics simulations. Special thanks go to Mr. Tony Chazirakis for his continuous support with but not limited to the MD-PAR package, his insight in parallelization and especially his help with the Alternative Derivation of the tail correction, see section (4.3). I would also like to thank Dr. Petra Bacova for her continuous support and guidance. Finally I am thankful to Dr. Anastasia Rissanou for providing me with the GROMACS configuration files and always being there to discuss questions.

Abstract

In this thesis we have studied molecular systems through molecular dynamics simulations by developing a new method to incorporate the tail correction potential energy in our calculations for an inhomogeneous system. We use molecular dynamics tools which are very important for predicting the structure properties as well as the computer design of materials. We demonstrate the importance of precise tail correction calculations and how otherwise they introduce a systematic error. We proceed to formulate an alternative derivation of the tail correction of the inhomogeneous system by direct integration. Our mathematical methods are written in C++ using MPI and openMP for parallelization in the MDPAR package. We test our implementations in MDPAR using two types of systems polyethylene with a frozen graphene layer and polyethylene in vacuum. Detailed results using comparisons of density histograms are are presented. We continue by testing the convergence of the tail correction potential energy.

Contents

List of Figures

List of Tables

1. Introduction

From the initial appearance of computers it was apparent how useful and necessary they were for solving mathematical problems. One could say that they acted as a catalyst in the rapid development of all sciences, which in return greatly influenced the development of computers themselves.

This boost provided the ability for the problems to become exponentially more complex and big – in such a scale that soon they exceeded the capabilities of the personal computers available at the time.

The solution came in the late 60s in the form of parallel computing, namely the usage of multi-core and later multi-processor systems, systems that combined multiple, usually similar, processors to distribute the load of the calculations. Such early examples were the D825 (MIMD) and the notoriously infamous – for the wrong reasons all together – ILLIAC IV (SIMD). When the project completed, after 11 years, it cost almost four times the original estimate, while was only one-quarter complete and underperformed compared to the then current systems [1].

Parallelization is a broad term. There are multiple types of parallelism (instruction-level parallelism, task parallelism etc), as well as different classes like multi-core computing, symmetric multiprocessing, distributed computing, cluster computing etc. Algorithmic methods have been developed to solve various problems: N-body problems, the Monte Carlo method, dense and sparse linear algebra to name a few.

Even though all these complex calculations can be broken down to the four basic arithmetical methods, in reality it is the pure volume of said calculations that makes even modern systems either unable to process or to process at extreme time or money cost.

The combination of arithmetical methods and parallel computation greatly diminishes the time required for given operations or in some cases even makes them possible – for example simulations that require enormous amounts of memory can not be supported by just one node.

Parallel computing is a very costly procedure. It requires preparation and a deep examination of the computational problem, experience in actual programming, time for testing and running the calculations and of course the other main factor is none other than the associated costs. Systems that can be used for parallel programming can be as small as few personal computers and laptops networked and as big as supercomputers that can cost up to hundreds of millions of euro in just infrastructure.

At this point we should note that parallelization is not a problem that can be solved just by "throwing money at it". There is absolutely no relation or guarantee that increasing the resources will linearly increase the performance. In truth problems may scale till a certain point after which there is no profit in increasing the number of cores used or might not be able to be parallelized at all.

The factor that affects mostly the scalability – supposing that the problem can be parallelized – turns up to be the quality of the code.

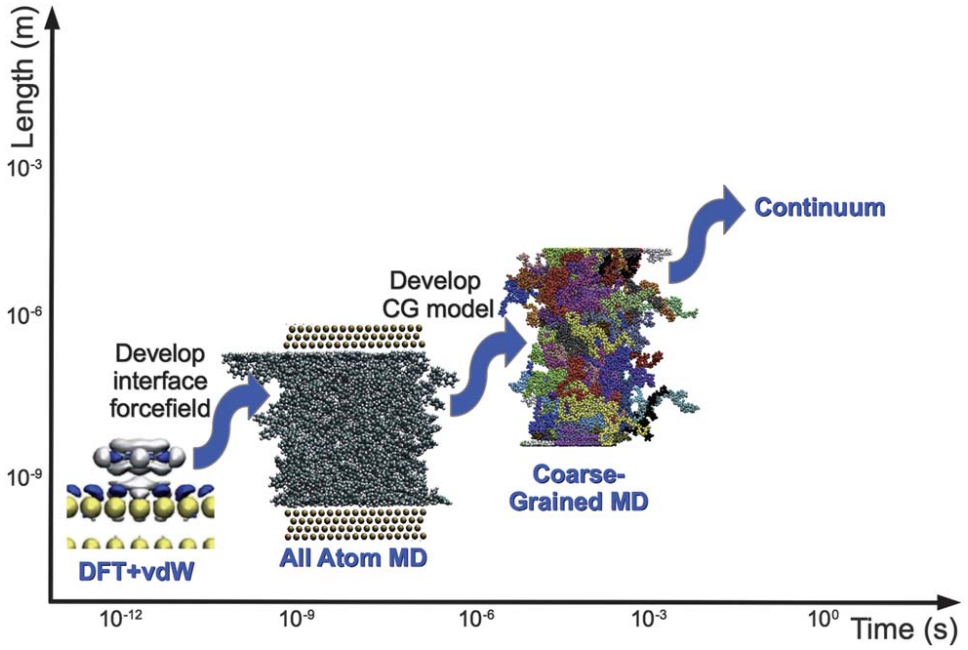

Figure 1.1: Multiscale modelling of polymer-solid interfaces from the electronic structure level, through the atomistic level, to the mesoscopic coarse-grained level and beyond [2].

Parallelization has seen excessive use in multiple calculations methods. One of them is molecular dynamics (MD) - a computer simulation

method for studying the physical movements of atoms and molecules but not limited to only those areas of research. A molecular dynamics simulation can be very expensive computationally-wise, as it is an N-body problem and the computational time scales with the number of atoms or species N in the system under study. Moreover, depending on the complexity of the studied system, the time and length scales in simulations can range from femtoseconds and Angstroms to units accessible by real experiments. All these time and lenght scales can not be covered by a single simulation technique, therefore we distinguish different types of simulations starting from the DFT (density functional theory) used for the quantum description of the system, to the atomistic level, through the mesoscopic coarse-grained level and up to the continuum level as seen in Fig. 1.1 [2].

To cover the demand created by this field multiple commercial packages have been created and have seen widespread use, e.g. GROMACS [3], LAMMPS [4], ESPRESSO++ [5] just to name a few of the most popular ones. As the problems to be solved became increasingly more complex the processing power required to solve them grew to such heights that the creation and use of supercomputers was necessary. Notable supercomputers in Europe include Switzerland's Piz Daint, PRACE in Barcelona, Spain as well as the new Greek supercomputer facility Aris in Athens.

Even with the aforementioned packages and the usage of supercomputers in certain cases we can be limited or restricted in the simulations we want to run. Namely, commercial packages for the most part act like black boxes, i.e. a code that the user cannot inspect or modify beyond the choices provided by the developer. In the cases where one needs to use an algorithm or function which is not included in the specific commercially available package, there is no choice but to modify the problem or the approach used to solve the problem. Also, as the user can not view the code used for the parallelization in many cases the run of the code can not be optimized. Furthermore, depending on the problem and / or the cluster's hardware / software, performance of the code may not scale properly without necessary, however not possible modifications.

We can easily see that there is a clear need for homemade (open-source), well designed packages that provide the same functionality as the commercial ones while also providing modularity, in the sense that one can modify or add features, as is required to tackle a specific problem. We aim in this work to overcome part of the above mentioned issues using the homemade MDPAR [6] package. Specifically, we will

implement in it our own algorithm which calculates the tail correction for an inhomogeneous system. This mathematical approach, correctly implemented takes into account the proper values of potential energy which are often substituted in commercially available packages by approximated values from the tail correction in homogeneous systems. We test the algorithm and report also the results obtained in the same solid / polymer and vacuum / polymer systems by commercially available GROMACS package.

2. N-body problem

2.1 Introduction to Classical MD Simulation

Molecular Dynamics (MD) is a computer simulation method for studying the physical movement of atoms and molecules (by computing the equilibrium and transport properties of a classical many body system). Or more commonly known, a N-body type of problem. With the term classical we imply that motion of the atoms or molecules obey the laws of classical mechanics.

An N-body problem normally predicts the individual motions of a group of celestial objects which interact with each other gravitationally. It can further be expanded to the microscopic plane by substituting celestial objects for atoms or molecules. Gravity is replaced by a potential.

MD simulations to some extend mimic real experiments. In a lab we need to prepare the sample and the correct measuring instrument. Similarly, in MD we start by setting up the initial configuration of the system and then we measure the property of interest during a certain time interval. Moreover, simulations bridge the gap between the experiments and the theory. They allow us to study properties not easily accessible through the experimental techniques and at the same time we can use the simulation results in order to test the theoretical predictions.

The procedure that we follow during the MD simulation can be summarized in distinct points and is usually referred to as the MD algorithm.

The general algorithm scheme consists of:

1. Initialization: we create the initial configuration of the model system that includes the initial coordinates and velocities for each particle.

2. Force calculations: in this point the forces acting on every particle are calculated.

3. Integration of the equations of motion: the evolution of the system is achieved by the integration of the equations of motion through a chosen mathematical method.

4. Obtaining information about the actual state of the system: the present coordinates and velocities together with data required for the further analysis (energy, density etc.) are stored.

5. Presentation of final output values.

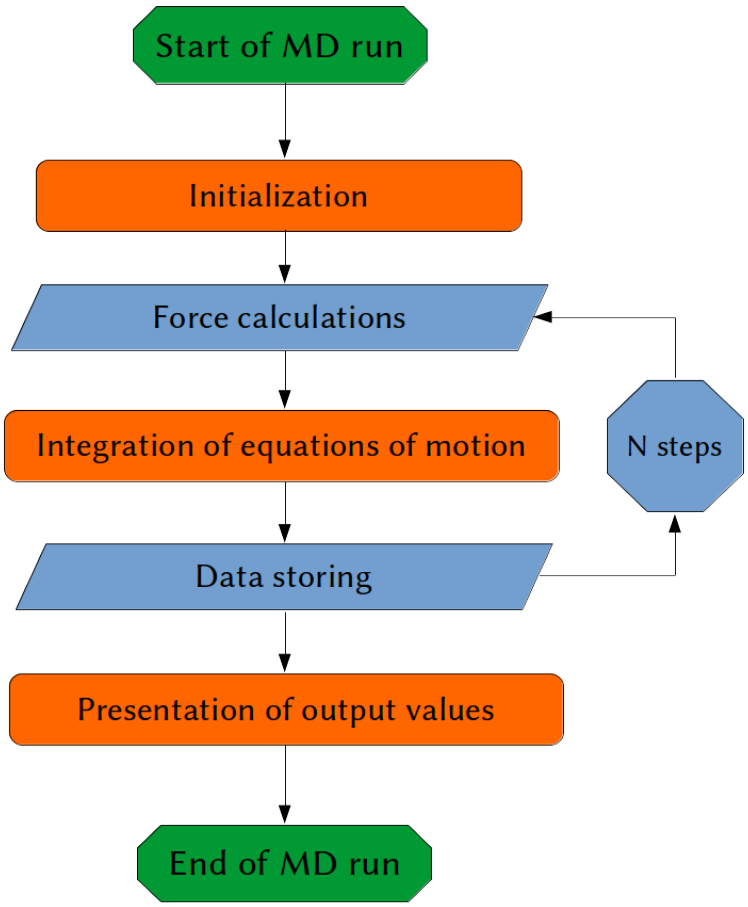

Figure 2.1: Graphical representation of the MD algorithm.

Points 2-4 are repeated till the system reaches a state of equilibrium. After that we perform a production run, starting the point 1 with the

equilibrated configuration. Again, we iterate the points 2-4 as long as it is required by the specific problem under investigation. Some problems may need longer iterations in order to obtain satisfactory results with the proper statistics.

In the following sections we will introduce more details concerning the Initialization (information about our model system can be found in section) and the algorithms related to the points 2 (section 2.2) and 3 (subsection 2.2.3) of the above mentioned scheme. We will pay particular attention to the potential energy calculation (the potential energy is introduced in section 3.2) where we will address the tail correction (TC) problem (see sections 4.1 and 4.2).

2.2 Force Calculation

In the molecular dynamics simulation the positions $\mathbf{r_i}$ and momenta $\mathbf{p_i}$ of the studied species are propagated by the equations of motion. The motion of the species is detected during the simulation and corresponds to the simulation trajectory.

2.2.1 Newton's equations of motion

The MD method follows Newton's second law of motion,

$$\mathrm{F_i} = \mathrm{m_i}\ddot{\mathrm{r}}_\mathrm{i} = \dot{\mathrm{p}}_\mathrm{i} \qquad \mathrm{i} = 1, 2, ..., \mathrm{N} \tag{2.1}$$

where F_i is the force exerted on particle i, m_i is the mass and $\ddot{r}_\mathrm{i}$ the acceleration of particle i. The single or double dot in the previous equation denotes the first and the second time derivative, respectively.

Let us assume that the particles in the system are interacting *via* given potential $V(\mathbf{r})$ where $\mathbf{r}$ represents the full set of coordinates. The force then can be written as the gradient of the potential:

$$F_\mathrm{i} = -\nabla_\mathrm{i} V \tag{2.2}$$

or

$$\mathrm{F_i} = -\frac{\partial}{\partial \mathrm{r_i}}\mathrm{V}(\mathbf{r}) \tag{2.3}$$

Combining 2.1 with 2.3 yields:

$$-\frac{\partial V}{\partial r_\mathrm{i}} = m_\mathrm{i}\ddot{r}_\mathrm{i} \tag{2.4}$$

2.2.2 Other formulations of classical equations

In this section we introduce two more formalisms to formulate the classical equations of motion namely, Lagrangian and Hamiltonian formalism.

For our system of N interacting molecules and potential V, we denote the generalized coordinates $\mathbf{q} = (\mathbf{q_1}, \mathbf{q_2}, ..., \mathbf{q_N})$ and their first time derivative $\dot{\mathbf{q}} = (\dot{\mathbf{q_1}}, \dot{\mathbf{q_2}}, ..., \dot{\mathbf{q_N}})$. In the Lagrange formulation, following Cartesian coordinates ($n = 3N$), the equations of motion can be written in the following form:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = 0 \qquad\qquad i = 1, 2, ..., N \tag{2.5}$$

where L is the Lagrangian of the system, defined as the difference between kinetic and potential energies:

$$L(\mathbf{q}, \dot{\mathbf{q}}, t) \equiv K(\dot{\mathbf{q}}) - V(\mathbf{q}) \tag{2.6}$$

By expanding the kinetic energy term we arrive to the expression:

$$L = \frac{1}{2}\sum_{i=1}^{n} m_i \dot{q_i}^2 - V(\mathbf{q}) \tag{2.7}$$

In the similar fashion, the generalized momenta is defined as:

$$p_i = \frac{\partial L}{\partial \dot{q}_i} \tag{2.8}$$

The second formulation we are going to mention is the Hamiltonian formalism. Using this formalism the equations of motion are described as:

$$\dot{q}_i = \frac{\partial H}{\partial q_i}, \dot{p}_i = -\frac{\partial H}{\partial q_i} \tag{2.9}$$

In the previous equation the H stands for the Hamiltonian function which is the sum of the kinetic and potential energies:

$$H(\mathbf{p}, \mathbf{q}) = \mathrm{K}(\dot{\mathbf{p}}) + \mathrm{V}(\mathbf{q}) \tag{2.10}$$

Hamilton's equations of motion are reversible in time, i.e. the evolution of the system can be retraced backwards.

2.2.3 Integration Algorithm

Since all forces between the particles have been computed, it is time consuming to integrate Newton's equations of motion. There are multiple algorithms designed to do this but we will use the so-called Verlet algorithm, which is not only one of the simplest, but also usually the best performing one.

What defines an algorithm as *good* or *bad* is how well it satisfies various criteria, some of them we will briefly discuss here:

- Usually one would instantly think of speed first but that is not always the case. Compared to other MD computations integration requires rather small computing time. Similarly, memory usage should be kept minimal but this is usually not an issue as modern systems have in their disposition much greater memory storage.
- Accuracy for large time steps is more important. The longer the time step we can use, the fewer evaluations of the forces are needed per unit of simulation time.
- Newton's equations of motion are time reversible and so should our algorithms be.
- Algorithm should satisfy the energy conservation law.

Assuming Cartesian coordinates with $\mathbf{r} = \{r_1, ..., r_N\}$ we start the derivation of the algorithm with a Taylor expansion of the coordinate of a particle around time t [7]:

$$\mathrm{r}(\mathrm{t}+\Delta\mathrm{t}) = \mathrm{r}(\mathrm{t}) + \nu(\mathrm{t})\Delta\mathrm{t} + \frac{\mathrm{f}(\mathrm{t})}{2\mathrm{m}}\Delta\mathrm{t}^2 + \frac{\Delta\mathrm{t}^3}{3!}\dddot{\mathrm{r}} + \mathrm{O}(\Delta\mathrm{t}^4) \tag{2.11}$$

$$\mathrm{r}(\mathrm{t}-\Delta\mathrm{t}) = \mathrm{r}(\mathrm{t}) - \nu(\mathrm{t})\Delta\mathrm{t} + \frac{\mathrm{f}(\mathrm{t})}{2\mathrm{m}}\Delta\mathrm{t}^2 - \frac{\Delta\mathrm{t}^3}{3!}\dddot{\mathrm{r}} + \mathrm{O}(\Delta\mathrm{t}^4) \tag{2.12}$$

Summing equations 2.11,2.12, we obtain:

$$\mathrm{r}(\mathrm{t}+\Delta\mathrm{t}) + \mathrm{r}(\mathrm{t}-\Delta\mathrm{t}) = 2\mathrm{r}(\mathrm{t}) + \frac{\mathrm{f}(\mathrm{t})}{\mathrm{m}}\Delta\mathrm{t}^2 + \mathrm{O}(\Delta\mathrm{t}^4)$$

or

$$\mathrm{r}(\mathrm{t}+\Delta\mathrm{t}) \approx 2\mathrm{r}(\mathrm{t}) - \mathrm{r}(\mathrm{t}-\Delta\mathrm{t}) + \frac{\mathrm{f}(\mathrm{t})}{\mathrm{m}}\Delta\mathrm{t}^2 \tag{2.13}$$

Verlet algorithm does not require the velocity to compute the new position. However, one, can derive the velocity from knowledge of the trajectory, using:

$$\mathrm{r}(\mathrm{t}+\Delta\mathrm{t}) - \mathrm{r}(\mathrm{t}-\Delta\mathrm{t}) = 2\nu(\mathrm{t})\Delta\mathrm{t} + \mathrm{O}(\Delta\mathrm{t}^3)$$

or

$$\nu(t) = \frac{r\,(t+\Delta t) - r\,(t-\Delta t)}{2\Delta t} + O(\Delta t^2) \tag{2.14}$$

2.3 Computational Methods for Molecular Dynamics

In this section we introduce mathematical methods widely used to either speed up the simulation algorithm or to avoid size effects.

2.3.1 Periodic Boundary Conditions

The objective of our simulation is to be able to predict properties of systems that are much bigger than our model system. Moreover our system is limited by the box dimensions and we have to be careful while considering phenomena happening in the boundaries of our box. This limitation can be overcome by employing a mathematical method which mimics the presence of infinite surrounding of our N-particle system. The method is based in the use of periodic boundary conditions.

Periodic boundary conditions are achieved by surrounding our simulation box with its exact copies throughout the space (Fig. 2.2a). During the course of the simulation any movement of a particle in the simulation box is precisely replicated by its periodic image in every one of the other boxes. If a particle exits the simulation box, one of its images will enter it from the opposite side (Fig. 2.2b). Thus in the direction where the periodic boundaries are applied there are no hard boundaries at the edge of the principal simulation box. In this way randomly chosen particle will interact with all other particles in this infinite periodic system, as well as with its own periodic image in all other cells. If we assume that the interactions are pairwise additive then the periodic boundary conditions will extend the calculations of the potential energy for infinite number of pairs. This general picture does not

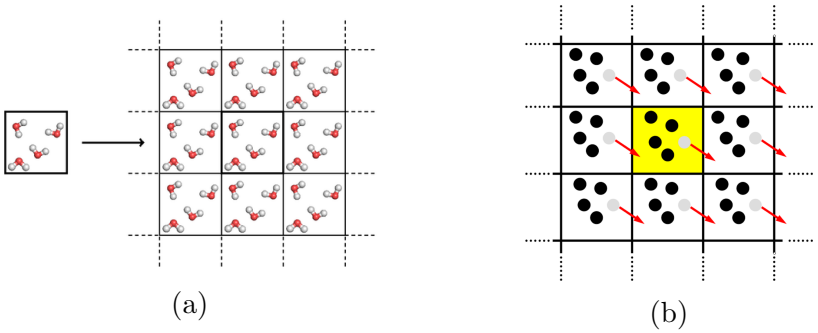

Figure 2.2: Periodic Boundary Conditions. (a) Our simulation space is surrounded by exact copies of the simulation box. (b) Movement inside the simulation box is replicated in all other boxes.

look particularly useful, however, in practice we deal with short-ranged interactions and the calculation is reduced only to the pairs inside of a certain cut-off distance r_{cut}. We will discuss the potential energy in Sec. 3.2 and how a specific choice of r_{cut} affects the results in Sec. 4.2.

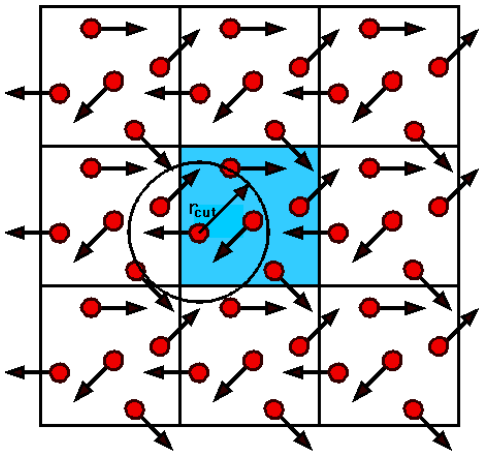

Figure 2.3: Minimum image convention

In Figure 2.3, the cut-off radius r_{cut} is schematically shown. Within this radius a given particle may interact with other particles placed in one of the neighbouring cells. In order to avoid a double calculation of the same pair positioned in various cells, we always calculate only the interaction with the image closest to the reference particle. This is known as the minimum image convention. We define r_{cut} to be $\leq \frac{L}{2}$, where L is the length of the simulation box.

2.3.2 Nose-Hoover Thermostat

In our simulations the number of particles, the volume and the temperature are kept constant. This setting is referred to in statistical mechanics as canonical ensemble or NVT. The canonical ensemble is the statistical ensemble that represents the possible states of a mechanical system in thermal equilibrium with a heat bath at a fixed temperature. The system can exchange energy with the heat bath, so that the states of the system will differ in total energy [8]. Once the system can be described as the canonical ensemble, one can use the expressions for the partition function derived by the statistical mechanics to compute the desired properties [7].

Several methods have been introduced to keep the temperature constant while using the micro-canonical ensemble. In our case, all of the systems will use the the Nosé–Hoover thermostat.

In one of the methods, the Andersen approach, the constant temperature is achieved by stochastic collisions with a heat bath [9]. The approach of Nosé is based on the use of an extended Lagrangian; that is, a Lagrangian that contains additional, artificial coordinates and velocities. To constrain temperature Nosé introduced an additional degree of freedom, s, in the Lagrangian, The parameter s plays the role of a heat bath whose aim is to damp out temperature deviations from the desired level. It requires adding to the total energy an additional potential term of the form:

$$V_s = gk_BTlns \tag{2.15}$$

and an additional kinetic energy term of the form:

$$K_s = \frac{Q}{2}\left(\frac{\dot{s}}{s}\right)^2 = \frac{p_s^2}{2Q} \tag{2.16}$$

In the above equations, g is the total number of degrees of freedom. In a system with constrained bond lengths, for example, $g = 3N_{atoms} - N_{bonds} - 3$, with N_{atoms} and N_{bonds} standing for the total number of atoms and bonds respectively. The value 3 subtracted in calculating g takes care of the fact that the total momentum of the simulation box is constrained to be zero by the periodic boundary conditions. Q and p_s represent the "effective mass" and momentum, respectively, associated with the new degree of freedom s. Equations of motion are derived from the Lagrangian of the extended ensemble, including the degree of

freedom s. Their final form, according to Hoover's analysis [10], is:

$$\dot{\mathbf{r}}_i = \frac{\mathbf{p_i}}{m_i} \tag{2.17}$$

$$\dot{\mathbf{p}}_i = -\frac{\partial V}{\partial \mathbf{r_i}} - \frac{\dot{s}}{s}\mathbf{p_i} \tag{2.18}$$

$$\dot{p}_s = \left(\sum_{i=1}^{N} \frac{\mathbf{p_i^2}}{\mathbf{m_i}} - g k_B T \right) / Q, \qquad p_s = Q\frac{\dot{s}}{s} \tag{2.19}$$

where the dot denotes a time derivative and p_i is the momentum of particle i.

An important result in Hoover's analysis is that the set of equations of motions is unique, in the sense that no other equations of the same form can lead to a canonical distribution. The total Hamiltonian of the system, which should be conserved during the MD simulation is:

$$H_{Nose-Hoover} = \sum_{i=1}^{N} \frac{\mathbf{p_i^2}}{\mathbf{m_i}} + V\left(\mathbf{r^N}\right) + g k_B T ln s + \frac{p_s^2}{2Q} \tag{2.20}$$

2.3.3 Neighbor Lists

In order to determine how many particles interact inside of the shell with radius r_{cut} one has to check all the possible pairs within the box. This fact may be the slow point of the simulation algorithm, as the number of pairs scales quadratically with the number of particles.

Since the cut-off radius ensures that a given particle will only interact with a finite number of other particles, we can build a list of all these particles. Such a list is known as a Verlet neighbour list. In Verlet lists each particle is surrounded by one additional shell with a radius r_l. The efficiency of the algorithm will depend on the thickness of the shell. Additionally, the list is updated every certain time and the proper estimation of the update frequency is crucial to the performance of the algorithm. Both of the values may be tuned according to the needs of the problem. For example, if we can estimate a time scale at which most of the particles participating in the pair interaction will not move distance greater than the r_{cut}, this time estimation can help us to find the most convenient parameters r_i and frequency.

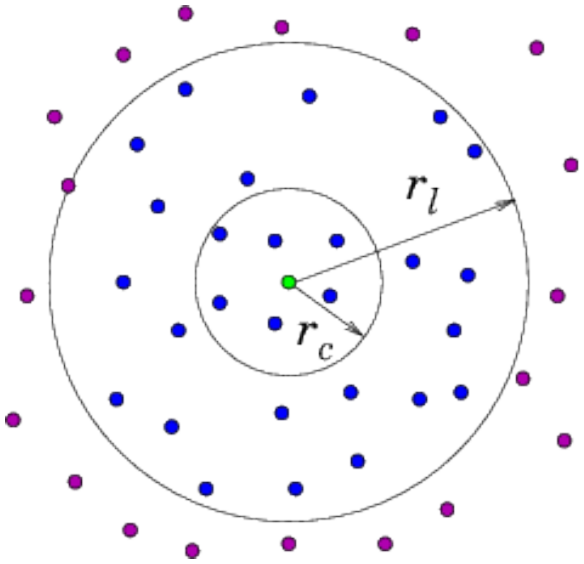

Figure 2.4: Neighbour List

2.3.4 Cut-off

The computational cost associated with molecular dynamics simulations is still very large, however representing a significant obstacle to the more widespread applications of MD simulation techniques to the study of complex biological processes [11]. MD simulations, like many other computational methods, face a trade off between computational efficiency and accuracy. Namely by reducing the quantity of computations in our simulations we are decreasing the accuracy we get.

In order to perform MD simulations less expensively or on longer timescales, a number of approximations of the potential energy function are often employed. The most computationally expensive part of MD simulation is generally the calculation of non-bonded forces, including both electrostatic and Van der Waals interactions, which act between all pairs of atoms. A common approach to reduce the cost of this computation is to set a maximum distance of interactions $R_{\mathrm{cut-off}}$, limiting the scope of interactions to a sphere with volume $\frac{4\pi}{3} r^3_{\mathrm{cut-off}}$ under the condition that potential degrades quickly enough in that distace, to ignore any interaction between atoms separated by more than the cut-off distance. This approach is generally accepted as being sufficiently accurate for Van der Waals forces, which decay rapidly to zero as the distance increases.

3. Model and simulation details

3.1 Simulated Systems

All simulations we run, are executed in the (NVT) canonical ensemble at 450K with the use of the Nose-Hoover thermostat, see section [2.3.2].

In the simulatiom molecules can be represented either by the atomistic or the coarse-grained models. The atomistic model takes into account all the atoms of a chain while the coarse grained model separates the chain into discrete atom groups. The coarse graining procedure reduces the number of degrees of freedoms which makes the simulation computationally more feasible, however we lose the information at the atomistic scale, e.g. partial charge, chemical affinity [12].

The atomistic model can either include all-atom details (explicit atom model) or it neglects the hydrogens (united atom model, UA). The explicit atom model considers each type of atom as independent spherical center of interaction, including hydrogen, in comparison to the united atom model UA which represents the atoms of hydrogen and carbon in one ethyl CH_2 or methyl CH_3 unit. In this representation carbon and hydrogens are considered as a unified center of interactions.

We used the united atom model for the polyethylene and graphene molecules. It gives us the opportunity to simulate systems in the atomistic scale without losing information about the chemical structure as it would be in the case of coarse grained model, however it is computationally cheaper than explicit atom model and we also avoid possible quantum effects coming from the movement of light hydrogen atoms. Including hydrogens would increase the number of atoms in the system and in the case of large systems it would lead to exponentially larger computational times.

system	number of graphene atoms	number of chains	number of chain monomers	box size [nm]
S1	2400	200	78	5.68 x 4.796 x 25.0
F1	0	200	78	5.68 x 4.796 x 120.0
F2	0	2000	78	11.36 x 9.592 x 120.0

Table 3.1: System Characteristics

3.1.1 System S1: Polyethylene chains with Graphene

Our first simulation system which we will refer to as "System S1" consists of polyethylene chains deposited on a graphene surface and confined by a hard wall. In our work we used the united-atom model in which the polymer chain was represented by the sequence of united atoms consisting of one carbon and two or three hydrogen atoms per unit. Namely, in our case the chain contains two CH3 groups interconnected by CH2 units.

Graphene is placed in the bottom of our simulation box and is formed by carbon atoms arranged in honey comb structure. The graphene layer is frozen, which means that during the simulation the position of the graphene atoms is constant.

The system is arranged in a way that is periodic only in two directions i.e. we use the periodic boundary conditions only in x and y direction while the z dimension is fixed.

The details related to the composition of the system S1 can be found in Table 3.1 and a typical snapshot is shown in Fig. 3.1.

3.1.2 System F1: Polyethylene system with 200 chains

Our second simulation system which we will refer to as "System F1" consists of polyethylene chains in vacuum. In this case no surface was present and the polyethelyne chains form a film surrounded from both sides by vacuum. Again we used the united-atom model and the system was periodic only in the x,y direction.

A representative snapshot is shown in Fig. 3.1b.

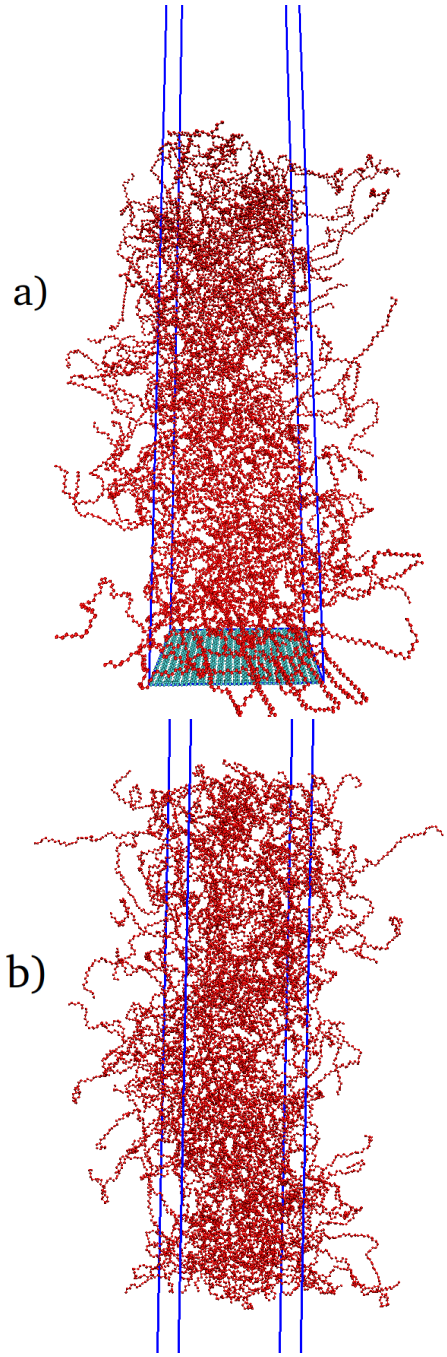

Figure 3.1: a) A typical snapshot from the simulation of S1 (top) and b) F1 system (bottom). Red beads represent the monomers of poly(ethylene), the cyan surface at the bottom is a graphene sheet and the blue lines are used to illustrate the boundaries of the box.

3.1.3 System F2: Polyethylene system with 2000 chains

Our third simulation system which we will refer to as "System F2" consists again of polyethylene chains in vacuum. It is similar to system F1 but this time the simulation box has been increased four times compared to F1, to allow for a higher maximum value of R_{cut} defined previously in Sec. 2.3.4.

Details for all the systems are summarized in Table 3.1.

3.2 Potential Energy

To start a MD simulation we need to set up initial parameters for the interactions between the particles in our systems. Assume a system described in $\mathbf{r} = \{r_1, ..., r_N\}$. The interactions are described by a potential defined as the derivative of the total force acted on all atoms of the system. We distinguish bonded and non-bonded interactions therefore the inter-atomic potential $V(\mathbf{r})$ consists of two parts:

$$V(\mathbf{r}) = V_{\text{bonded}}(r) + V_{\text{non-bonded}}(r) \tag{3.1}$$

$V_{\text{bonded}}(r)$ stands for the bonded potential and $V_{\text{non-bonded}}(r)$ is the potential corresponding to the non-bonded interactions. The potential is pair-wise, it means that it only depends on the distance r between a pair of particles. The description of these interactions constitutes a force field. The force field parameters needed for a certain system can be usually found in the literature being derived from quantum simulations. To model the polyethylene chain we used well-established TraPPE force field, widely used for alkanes [13]. In the following section using the united-atom model (UA) we will introduce the expressions for the components of the potential and we will report the used set of parameters.

Bonded Potential Energy

The potential energy corresponding to bonded interactions can be written as a sum of three terms:

$$V_{\text{bonded}}(r) = V_{\text{bonds}}(r) + V_{\text{angles}}(r) + V_{\text{dihedrals}}(r) \tag{3.2}$$

where V_{bonds} is the potential describing the interaction due to the bonds between the particles, V_{angles} is the angle potential among three con-

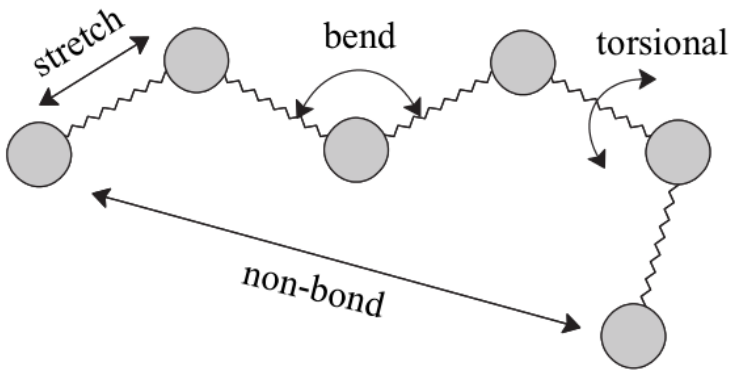

Figure 3.2: Types of particle interactions.

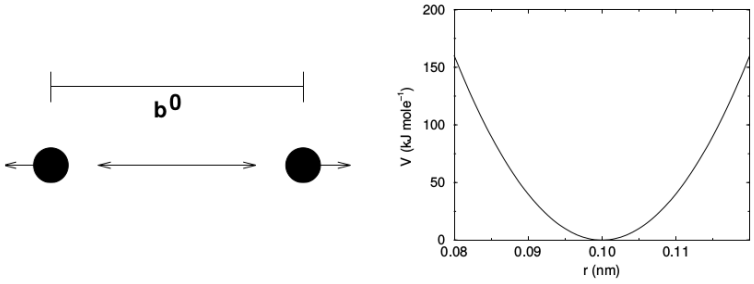

Figure 3.3: Bond stretching (left) and bond stretching potential (right)

secutive particles in the chain, $V_{\text{dihedrals}}$ is the potential describing the interaction among 4 consecutive atoms i.e., is the angle ϕ of the two planes defined from the four consecutive particles.

Bond Potential:

If there are bonds along consecutive atoms, then the bonding distance between atoms $i, i+1$ is defined as $b = |r_i - r_{i+1}|$, where r_i, r_{i+1} are the positions of the atoms i and $i+1$, respectively.

In such case we can express harmonic bonding potential $V_{\text{bond}}(r)$ as:

$$V_{\text{bonds}}(r) = \sum_{\text{bonds}} k_b \left(b - b_0\right)^2 \tag{3.3}$$

where V_{bonds} is the potential energy function for stretching a bond between the atoms i and $i+1$, k_b is the harmonic force constant and b_0 is the equilibrium bond length. The harmonic function is the simplest and sufficient form for determining most equilibrium geometries.

atom group	b_0 [nm]	k_b [kJ/mol*nm^2]
CH_2 CH_2	0.154	83736.0
CH_2 CH_3	0.154	83736.0
CH_3 CH_2	0.154	83736.0

Table 3.2: Parameters used in bond potential[13]

A sketch of a bond stretching together with a graphical representation of the potential can be seen in Fig. 3.3.

The parameters k_b and b_0 applied in the simulation of polyethylene are summarized in Tab. 3.2.

Angle Potential:

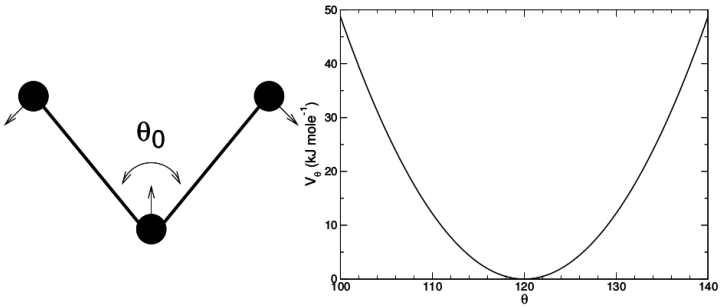

Figure 3.4: Sketch of angle vibration (left) and bond angle potential (right)

The functional form for the angle potential V_{angles} can be written as:

$$V_{\text{angles}} = \sum_{\text{angles}} k_\theta \left(\theta - \theta_0\right)^2 \tag{3.4}$$

where k_θ is the angle bending force constant and θ is the angle between successive bonds. Let us define the bond vector $\mathbf{b_i} = \mathbf{r_i} - \mathbf{r_{i+1}}$. Consequently, the θ angle is calculated by the formula:

$$\cos\theta = \frac{\mathbf{b_i}\mathbf{b_{i+1}}}{|\mathbf{b_i}||\mathbf{b_{i+1}}|} \tag{3.5}$$

A sketch of a angle vibration together with a functional form of the potential can be seen in Fig. 3.4.

atom group	$\theta_0(deg)$	k_θ [kJ/mol*deg^2]
$CH_2\ CH_2\ CH_2$	114	519.611
$CH_2\ CH_2\ CH_3$	114	519.611
$CH_3\ CH_2\ CH_2$	114	519.611

Table 3.3: Parameters used in angle potential[13]

To calculate the angle potential we used parameters given in Tab. 3.3.

Dihedral Potential:

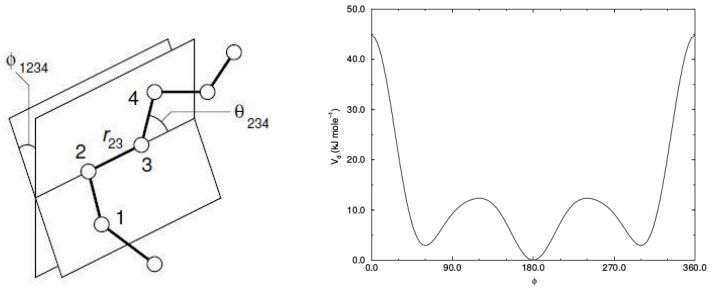

Figure 3.5: Left: Geometrical representation of the planes in the chain. r_{23} is the diatomic distance and angle ϕ_{1234} is the dihedral angle used in 3.6. Right: Ryckaert-Bellemans dihedral potential [14]

In order to describe the dihedral potential we use the Ryckaert-Bellemans function given by the form:

$$V_{\text{dihedrals}} = \sum_{n=0}^{5} C_n \left(\cos\left(\phi - \pi\right)\right)^n \tag{3.6}$$

where ϕ is the angle between the planes defined as following. If we imagine a simple polymer chain and draw it in a way as it is illustrated in Fig. 3.5 we can define the angle ϕ_{1234} as:

$$\cos\phi = -\frac{\left(\mathbf{b_{12}} \times \mathbf{b_{23}}\right)\left(\mathbf{b_{23}} \times \mathbf{b_{34}}\right)}{|\mathbf{b_{12}} \times \mathbf{b_{23}}||\mathbf{b_{23}} \times \mathbf{b_{34}}|} \tag{3.7}$$

where $\mathbf{b_{ij}}$ is the bond vector between atoms i, j.

A conversion from the GROMACS convention to the MDPAR can be achieved by multiplying every coefficient C_n by $(-1)^n$ as described in chapter 4.2.13 of the GROMACS manual [3].

atom group	c_0	c_1	c_2	c_3	c_4	c_5 [kJ/mol]
$CH_3\ CH_2\ CH_2\ CH_2$	9.276	12.154	-13.117	-3.058	26.238	-31.493
$CH_2\ CH_2\ CH_2\ CH_2$	9.276	12.154	-13.117	-3.058	26.238	-31.493
$CH_2\ CH_2\ CH_2\ CH_3$	9.276	12.154	-13.117	-3.058	26.238	-31.493

Table 3.4: Parameters used in dihedral potential[13]

The Ryckaert-Bellemans potential is plotted in Fig. 3.5.

The parameters used to calculate the dihedral potential are given in Tab. 3.4.

Non-bonded Potential Energy

The non-bonded potential is written as a sum of two terms the Van der Waals and the electrostatic interactions:

$$V_{\mathrm{non-bonded}}(r) = V_{\mathrm{LJ}}(r) + V_{\mathrm{c}}(r) \tag{3.8}$$

Thus, non-bonded interactions which are dependent of the distance between the particles and are the sum of Van der Waals described by a typical Lennard-Jones (LJ) potential and electrostatic by Coulomb potential.

Lennard-Jones Interactions

The Lennard-Jones potential V_{LJ} between two atoms i and j equals:

$$V_{\mathrm{LJ}} = 4\epsilon_{ij}\left(\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right) \tag{3.9}$$

ϵ_{ij} is the depth of the LJ well for a distance of $r_{\mathrm{ij}} = r_m = 2^{\frac{1}{6}}\sigma_{\mathrm{ij}}$, σ_{ij} is the finite distance on which the inter-particle potential is zero. For the calculation of σ_{ij} and ϵ_{ij} parameters we can use the Lorent-Berthelot combination rules:

$$\sigma_{ij} = \frac{1}{2}\left(\sigma_i + \sigma_j\right), \epsilon_{ij} = \sqrt{\epsilon_i \epsilon_j} \tag{3.10}$$

The parameters σ and ϵ for each atom are coming either from fits to experimental data or quantum-mechanical calculations. For our simulations their values can been seen in the table below:

type	σ [nm]	ϵ [kJ/mol]
CH_2	0.395	0.3824
CH_3	0.395	0.3824
C_{GR}	0.340	0.2327

Table 3.5: Parameters σ and ϵ used in the Lennard-Jones potential.

The repulsive part of the Lennard-Jones equation (first term) describes the Pauli repulsion in small distances due to the overlap of the electron orbitals. The second term describes the attraction of particles in large distances due to Wan der Waals forces. The Lennard-Jones potential is a mathematically simple model potential and as such is used extensively in molecular simulations.

The Lennard-Jones potential is represented in Fig. 3.6 where the characteristic distances r_m and r_{cut} are also shown. The values of σ and ϵ for the PE model (CH_2 and CH_3) groups and the graphene atoms (C_{GR}) are shown in Table 3.5.

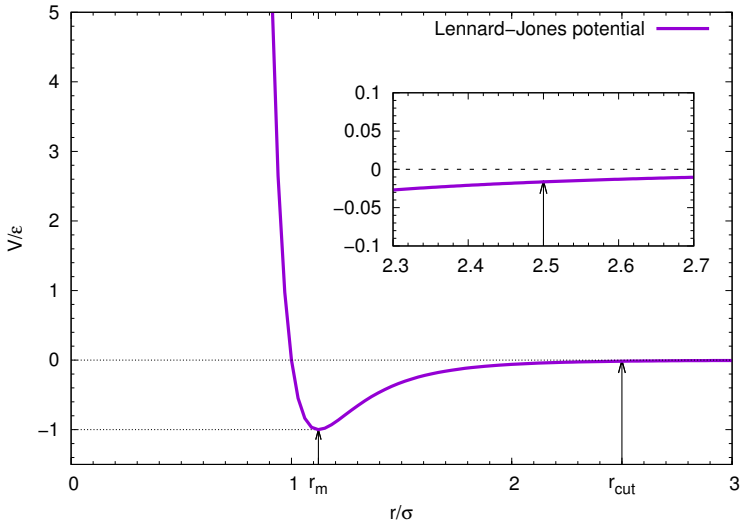

Figure 3.6: Lennard-Jones for a non-bonded potential. r_m is the distance at which the potential reaches its minimum value, r_{cut} being the cut-off distance.

Coulomb Interactions

If atom charges are present, we add the appropriate Coulomb po-

tential, which is the potential due to electrostatic interactions between two charged particles:

$$V_{\text{coulomb}}(r_{\text{ij}}) = \frac{Q_i Q_j}{4\pi\epsilon_0 r_{\text{ij}}} \tag{3.11}$$

where Q_i, Q_j are the charges of the particles i, j. r_{ij} is the distance between particles i, j and ϵ_0 is the dielectric constant in vacuum which equals to 8.854×10^{-12}C/mV. A schematic representation of all particle interactions is shown in Fig. 3.2.

In all our simulations the charges Q_i, Q_j are equal to zero which means that the calculation of the electrostatic energy can be omitted and so the non-bonded energy will have only one component, namely the Lennard-Jones potential.

3.3 Simulation Details

All the simulations were performed either by commercially available GROMACS package [3] or by home-made code MDPAR [6]. These two simulation programs differ in format of the configuration file and used units.

Units in GROMACS are nanometers in length, picoseconds for time, unified atomic mass for mass, kJ/mol for energy and Kelvins for temperature. On the other side MDPAR works with units used in LAMMPS package [4], namely Å, femtoseconds, gram/mol, Kcal/mol and Kelvins. For the sake of clarity unless otherwise specified all the data in the thesis are presented in the GROMACS units. As we mentioned in the previous chapter the simulation procedure is divided into two parts, equilibration and production run. Started with the initial configuration which included bulk polyethylene and graphene surface in the bottom in the case of S1 and only the polyethylene bulk in the case of F1. We let the systems equilibrate for approximately 0.5ns. We considered the system to be equilibrated when the system properties became independent of time. We start the production run after the equilibration. The length of the production run was approximately 2ns and the time step was 1fs. We saved the configuration every 2ps.

4. Tail Correction

Let us now consider a case where we perform a simulation of a system where the caclulated interactions are mostly short-range i.e, we ignore interactions of atoms which are further apart than a specific distance r_{cut}. In this context, short-ranged means that the total potential energy of a given particle i is dominated by interactions with neighboring particles that are closer than a short cutoff distance r_{cut}. The error that results when we ignore interactions with particles at larger distances can be made arbitrarily small by choosing r_{cut} sufficiently large. However, if we use periodic boundary conditions 2.3.1, the case that r_{cut} has to be less than $\frac{L}{2}$ (half the side of the periodic box) is of special interest because in that case we must consider the interaction of a given particle i only with the nearest periodic image of any other particle j. For periodic image convention see 2.3.1. If the intermolecular potential is not rigorously zero for distances $r > r_{cut}$, truncation of the intermolecular interactions at r_{cut} will result in a systematic error in total potential energy. If the intermolecular interactions decay rapidly, one may correct for the systematic error by adding a tail contribution to V^{total} [7]:

$$V^{\mathrm{total}} = \sum_{\mathrm{i<j}} V_{\mathrm{c}}(r_{\mathrm{ij}}) + \frac{N\rho}{2} \int_{\mathrm{r_{cut}}}^{\infty} \mathrm{drV(r)4\pi r^2} \tag{4.1}$$

$$V^{\mathrm{total}} = \begin{cases} 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] & r < r_{\mathrm{cut}} \\ 2\pi\rho \int_{r_{\mathrm{cut}}}^{\infty} \mathrm{drr^2V(r)} & r \geq r_{\mathrm{cut}} \end{cases} \tag{4.2}$$

where V_{c} stands for the truncated potential energy function for example LJ potential as shown in eq. 4.2 and ρ is the average number density and N is the number of particles interacting with the given potential. In writing down this expression, it is implicitly assumed that the radial distribution function $g(r) = 1$ for $r > r_{cut}$. Clearly, the nearest periodic image convention can be applied only if the tail correction is small. From equation 4.1 it can be seen that the tail correction to the potential

energy is infinite unless the potential energy function $V(\mathrm{r})$ decays more rapidly than r^{-3} (in three dimensions). This condition is satisfied if the long-range interaction between molecules is dominated by dispersion forces. Dispersion forces are part of the van der Waals forces and are a weak intermolecular force arising from quantum-induced instantaneous polarization multi-poles in molecules. They can therefore act between molecules without permanent multi-pole moments. However, for the very important case of Coulomb 3.11 and dipolar interactions, the tail correction diverges and hence the nearest-image convention cannot be used. In such cases, the interactions with all periodic images should be taken into account explicitly.

Several factors make truncation of the potential a difficult process. First of all, one should realize that although the absolute value of the potential energy function decreases with inter-particle separation r, for sufficiently large r, the number of neighbors is a rapidly increasing function of r. In fact, the number of particles at a distance r of a given atom increases asymptotically as $r^{\mathrm{d}-1}$, where d is the dimensionality of the system.

4.1 Tail Correction for isotropic system

Let us consider a specific example of isotropic system. For the three-dimensional system interacting by LJ, the pair potential in which atoms interact via pair potentials is given by:

$$V_{\mathrm{lj}} = 4\epsilon \left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} \right] \tag{4.3}$$

The average potential energy (in $3D$) of any given atom i is given by:

$$V_{\mathrm{i}} = (\frac{1}{2}) \int_0^\infty \mathrm{dr} 4\pi \mathrm{r}^2 \rho(\mathrm{r}) \mathrm{V}(\mathrm{r}), \qquad \mathrm{i} = 1, ..., \mathrm{N} \tag{4.4}$$

where $\rho(r)$ denotes the average number density at a distance r from a given atom i. The factor (1/2) takes care of double counting of pair interactions. If we truncate the potential at a distance r_{c}, we ignore for each atom the tail contribution V^{tail}:

$$V^{\mathrm{tail}} = (\frac{1}{2}) \int_{r_{\mathrm{c}}}^\infty dr 4\pi r^2 \rho(r) V(r) \tag{4.5}$$

where we have dropped the subscript i, because in an isotropic system all the atoms are identical. We assume that for $r \geq r_c$, the density $\rho(r)$ is equal to the average number density ρ. If $V(r)$ is the Lennard-Jones potential, we find for V^{tail}

$$\begin{aligned} V^{\text{tail}} &= \frac{1}{2} 4\pi\rho \int_{r_c}^{\infty} \text{drr}^2 \text{V(r)} \\ &= \frac{1}{2} 16\pi\rho\epsilon \int_{r_c}^{\infty} \text{drr}^2 \left[\left(\frac{\sigma}{\text{r}}\right)^{12} - \left(\frac{\sigma}{\text{r}}\right)^{6} \right] \\ &= \frac{8}{3}\pi\rho\epsilon\sigma^3 \left[\frac{1}{3} \left(\frac{\sigma}{r_c}\right)^{9} - \left(\frac{\sigma}{r_c}\right)^{3} \right] \end{aligned} \tag{4.6}$$

As an example, if we assume values of r_c= 2.5σ and density $\rho\sigma^3$= 1, we find V^{tail}= -0.535ϵ, where ϵ= 0.0914 kcal/mol and ϵ = 3.95 Å (TRaPPE model [13]). Therefore, for N atoms the total truncated energy is $\text{N} \cdot \text{V}^{\text{tail}} = \text{N}(-0.535\epsilon)$.

4.2 Tail Correction for inhomogeneous system

In a many-body, inhomogeneous system with more than one type of atoms the contribution to the long-range correction in the potential energy, V_{tails}, will be the sum of the corrections from all atoms (or species):

$$V_{\text{tail}} = \sum_{\text{i}=1}^{N} V_{\text{tail,i}} \tag{4.7}$$

where n denotes the number of molecules and $V_{\text{tail,i}}$ is the correction in energy for molecule i due to its interaction with all other molecules.

In turn $V_{\text{tail,i}}$ is the sum of corrections in energy due to the interaction of the $i-$th molecule with molecules of all molecule types:

$$V_{\text{tail,i}} = \sum_{\text{t}_\text{j}=1}^{\text{t}_\text{n}} V_{\text{tail}} \left[\text{t}_\text{i}, \text{t}_\text{j}\right] \tag{4.8}$$

where t_n is the number of molecule types, t_j is the type of molecule j and t_i is the type of the $i-$th molecule. Term in the brackets shows the dependence of the correction on the type of i, j.

We will explain the above relations on a specific example: Let us consider a system that has three atom types, namely C,H,O. Thus t_n will be equal to 3. t_j will iterate between the 3 different types, so t_1 will represent the atom type of C, t_2 the atom type of H and finally t_3 the atom type of O.

For a pair of atoms in a system that is periodic in the x, y axis the correction is given by the following equation presented in ref. [15]:

$$V_{\text{tail}}\left[\mathrm{t_i}, \mathrm{t_j}\right] = \int_{z_j}^{z_j+\delta} dz \int_{\varrho}^{\infty} \mathrm{rdr} \int_{0}^{2\pi} \mathrm{d}\varphi \frac{1}{\mathrm{L_x L_y}\delta} \mathrm{U_{t_i,t_j}}[\sqrt{\mathrm{r}^2 + (\mathrm{z} - \mathrm{z_i})^2}] \tag{4.9}$$

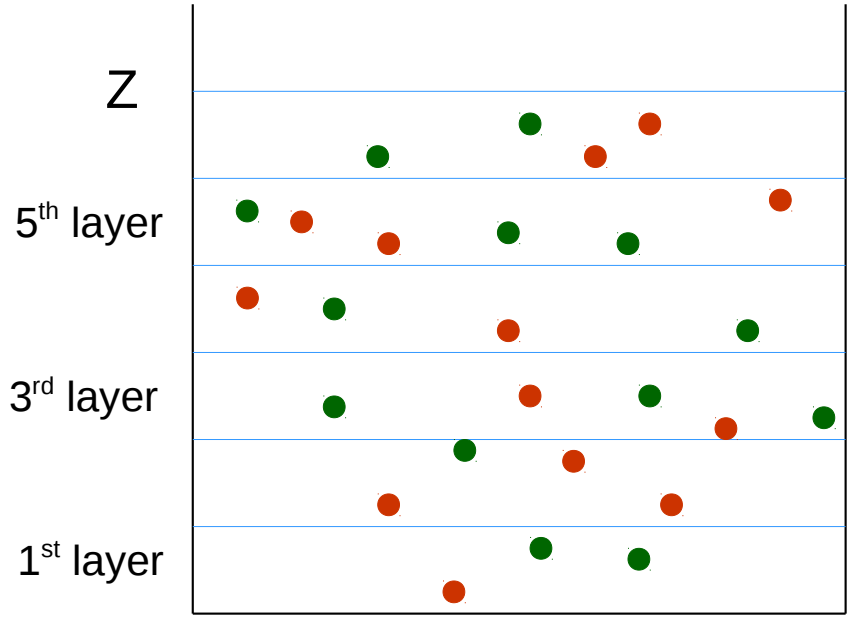

Figure 4.1: Our simulation box divided in J_{layers} along the z direction

A further approach is the division of the non periodic axis in our case z in discrete layers, each of thickness δ. We follow this approach and we refer to them as J_{layer} see Fig. 4.1. Taking advantage of the symmetry with respect to the polar angle φ after the integration we arrive to an expression:

$$V_{\mathrm{tail}}\left[\mathrm{t_i}, \mathrm{t_j}\right] = \frac{1}{2} 2\pi \sum_{\mathrm{J_{layer}}=0}^{\mathrm{N_b}-1} N_{t_j}\left(J_{layer}\right) \times \int_{\varrho}^{\infty} \frac{1}{L_x L_y} U_{t_i,t_j}\left[\tilde{r}\right] \mathrm{rdr} \qquad (4.10)$$

$$\tilde{r} = \sqrt{r^2 + (J_{layer}\delta - z_i)^2}$$

where we sum over all the layers, J_{layer}, ranging from 0 till $N_b - 1$, N_b being the number of layers. N_{t_j} stands for the number of particles of type j per layer. U_{t_i,t_j} represents the interaction of a molecule of type i with a molecule of type j. The factor $\frac{1}{2}$ takes care of the duplicated pair iterations. L_x, L_y are the dimensions of plane x,y respectively. The lower limit ϱ of the integral is the minimum distance we integrate over given by:

$$\varrho = \sqrt{max\{R_c^2 - (J_{\mathrm{layer}}\delta - z_i)^2, 0\}} \qquad (4.11)$$

R_c denotes the cut off distance, while z_i is the coordinate of molecule **i** on the z axis.

By combining equations 4.7, 4.8, 4.10 we get the complete equation:

$$V_{\mathrm{tail}} = \sum_{\mathrm{i}=1}^{n} \sum_{\mathrm{t_j}=1}^{\mathrm{t_n}} \sum_{\mathrm{J_{layer}}=0}^{\mathrm{N_b}-1} \pi N_{t_j}\left(J_{layer}\right) \times \int_{\varrho}^{\infty} r \frac{1}{L_x L_y} U_{t_i,t_j}\left[\tilde{r}\right] \mathrm{dr} \qquad (4.12)$$

By substituting U_{t_i,t_j} with the common expression of Lennard-Jones in 4.10 we end up with:

$$V_{\text{tail}} = \sum_{\text{i}=1}^{n} \sum_{\text{t}_\text{j}=1}^{\text{t}_\text{n}} \sum_{\text{J}_\text{layer}=0}^{\text{N}_\text{b}-1} \frac{\pi N_{t_j}\left(J_{layer}\right)}{L_x L_y} \times \int_{\varrho}^{\infty} 4\epsilon_{t_i,t_j} r \left(\frac{\sigma_{t_i,t_j}^{12}}{\tilde{r}^{12}} - \frac{\sigma_{t_i,t_j}^{6}}{\tilde{r}^{6}} \right) dr \tag{4.13}$$

Solving the integration we arrive at:

$$\begin{aligned}
&\int_{\varrho}^{\infty} 4\epsilon_{t_i,t_j} \left(\frac{r\sigma_{t_i,t_j}^{12}}{\left(r^2 + \left(J_{layer}\delta - z_i\right)^2\right)^6} - \frac{r\sigma_{t_i,t_j}^{6}}{\left(r^2 + \left(J_{layer}\delta - z_i\right)^2\right)^3} \right) dr \\
&= 4\epsilon_{t_i,t_j} \left(-\frac{\sigma_{t_i,t_j}^{12}}{10\left(r^2 + \left(J_{layer}\delta - z_i\right)^2\right)^5} + \frac{\sigma_{t_i,t_j}^{6}}{4\left(r^2 + \left(J_{layer}\delta - z_i\right)^2\right)^2} \right)_{\rho}^{\infty} \\
&= 4\epsilon_{t_i,t_j} \left(\frac{\sigma_{t_i,t_j}^{12}}{10\left(\rho^2 + \left(J_{layer}\delta - z_i\right)^2\right)^5} - \frac{\sigma_{t_i,t_j}^{6}}{4\left(\rho^2 + \left(J_{layer}\delta - z_i\right)^2\right)^2} \right) \\
&= \epsilon_{t_i,t_j} \left(\frac{2}{5} \frac{\sigma_{t_i,t_j}^{12}}{\left(max\{R_c^2 - (J_{\text{layer}}\delta - z_i)^2, 0\} + \left(J_{layer}\delta - z_i\right)^2\right)^5} - \right. \\
&\qquad \left. - \frac{\sigma_{t_i,t_j}^{6}}{\left(max\{R_c^2 - (J_{\text{layer}}\delta - z_i)^2, 0\} + \left(J_{layer}\delta - z_i\right)^2\right)^2} \right) \\
&= \epsilon_{t_i,t_j} \left(\frac{2}{5} \frac{\sigma_{t_i,t_j}^{12}}{{r_1}^{10}} - \frac{\sigma_{t_i,t_j}^{6}}{r_1^4} \right)
\end{aligned} \tag{4.14}$$

Substituting (4.14) in (4.13) we end with the final form of the equation:

$$V_{\text{tail}} = \sum_{\text{i}=1}^{n} \sum_{\text{t}_\text{j}=1}^{\text{t}_\text{n}} \sum_{\text{J}_\text{layer}=0}^{\text{N}_\text{b}-1} \frac{\pi N_{t_j}\left(J_{layer}\right)}{L_x L_y} \epsilon_{t_i,t_j} \left(\frac{2}{5} \frac{\sigma_{t_i,t_j}^{12}}{{r_1}^{10}} - \frac{\sigma_{t_i,t_j}^{6}}{r_1^4} \right) \tag{4.15}$$

$$r_1^2 = max\{R_c^2 - (J_{\text{layer}}\delta - z_i)^2, 0\} + (J_{\text{layer}}\delta - z_i)^2 \tag{4.16}$$

Since the main input in our calculations is the force, we just have to find the derivative according to $F = -\frac{dV}{dr_1}$ while keeping in mind that due to 4.16 we derive over z_i, thus the total force is:

$$F_z = \begin{cases} 0 & |J_{\text{layer}}\delta - z_i| \leq R_c \\ \sum_{\text{i}=1}^{n} \sum_{\text{t}_\text{j}=1}^{\text{t}_\text{n}} \sum_{J_{\text{layer}}=0}^{N_b-1} 2\epsilon_{t_i,t_j} \frac{\pi N_{t_j}\left(J_{\text{layer}}\right)}{L_x L_y} \left(-\frac{20}{5} \frac{\sigma_{t_i,t_j}^{12}}{\left(J_{\text{layer}}\delta - z_i\right)^{11}} - \right. & \\ \left. - (-4) \frac{\sigma_{t_i,t_j}^{6}}{\left(J_{\text{layer}}\delta - z_i\right)^{5}} \right) & |J_{\text{layer}}\delta - z_i| > R_c \end{cases}$$

$$= \begin{cases} 0 & |J_{\text{layer}}\delta - z_i| \leq R_c \\ \sum_{\text{i}=1}^{n} \sum_{\text{t}_\text{j}=1}^{\text{t}_\text{n}} \sum_{J_{\text{layer}}=0}^{N_b-1} 8\epsilon_{t_i,t_j} \frac{\pi N_{t_j}\left(J_{\text{layer}}\right)}{L_x L_y} \left(\frac{\sigma_{t_i,t_j}^{6}}{\left(J_{\text{layer}}\delta - z_i\right)^{5}} - \right. & \\ \left. - \frac{\sigma_{t_i,t_j}^{12}}{\left(J_{\text{layer}}\delta - z_i\right)^{11}} \right) & |J_{\text{layer}}\delta - z_i| > R_c \end{cases} \tag{4.17}$$

Previously when explaining the V_{tail} in eq. 4.10 we mentioned that the term $\frac{1}{2}$ takes care of not counting doubly the interactions between two pairs i, j. While that is correct for calculating the total potential energy, when calculating the force for an inhomogeneous system we explicitly have to take into account all the interactions of the species. Therefore, a factor of two is added in the above equations of force 4.17.

Respectively, the force on molecule $\boldsymbol{i}$, will be:

$$F_z^i = \begin{cases} 0 & |J_{\text{layer}}\delta - z_i| \leq R_c \\ \sum_{\text{t}_\text{j}=1}^{\text{t}_\text{n}} \sum_{J_{\text{layer}}=0}^{N_b-1} 8\epsilon_{t_i,t_j} \frac{\pi N_{t_j}\left(J_{\text{layer}}\right)}{L_x L_y} \left(\frac{\sigma_{t_i,t_j}^{6}}{\left(J_{\text{layer}}\delta - z_i\right)^{5}} - \right. & \\ \left. - \frac{\sigma_{t_i,t_j}^{12}}{\left(J_{\text{layer}}\delta - z_i\right)^{11}} \right) & |J_{\text{layer}}\delta - z_i| > R_c \end{cases} \tag{4.18}$$

Finally we may wish to consider the use of other potentials apart from the Lennard-Jones. Thus our equation would be updated to a more generic form:

$$F_z^i = \sum_{\text{t}_\text{j}=1}^{\text{t}_\text{n}} \sum_{J_{\text{layer}}=0}^{N_b-1} \frac{2\pi N_{t_j}\left(J_{\text{layer}}\right)}{L_x L_y} \times \left(-\frac{dV}{dr_1} \right) \left(\int_{\varrho}^{\infty} \left(D_{t_i,t_j}[\tilde{r}] \right) r dr \right) \tag{4.19}$$

4.3 Alternative Derivation of the Tail Correction for an inhomogeneous system

In this section we will present an alternative derivation of the force due to tail correction based on a direct integration on a two-dimensional disk. We follow closely the mathematically machinery published in this paper[15] but we implement a new approach derived by us. Instead of partitioning our simulation space with a periodicity in x,y direction and certain finite thickness in z direction, so to speak 3D layers of thickness δ we consider them 2D.
We assume the following points:

- The force exerted on a pair of particles is only dependent on their distance on axis z. With other words, since the system is homogeneous in x and y directions the tail correction energy along x,y is constant; thus the force in these directions is zero
- The thickness of a layer is in the order of a few Angstroms, so we can consider the height of a layer in respect to axis z to be equal to the middle of the layer's height

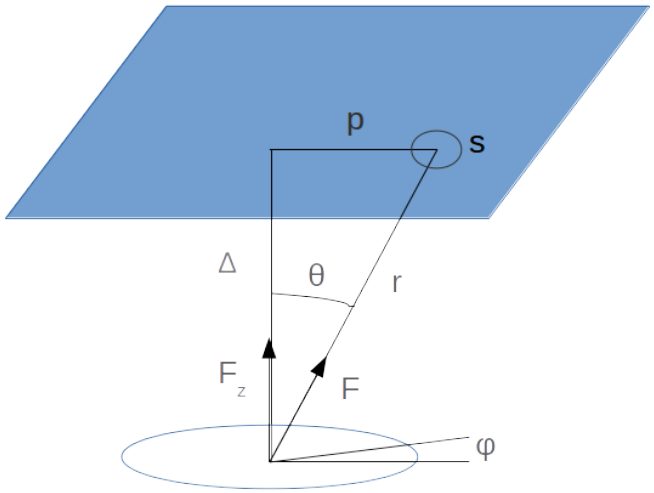

Figure 4.2: Representation of the distance between one particle and and the L_x, L_y plane.

$$\Delta = J_{layer} * \delta - z_i$$
$$r = \left(\rho^2 + \Delta^2\right)$$
$$F_z(s) = F * \cos\theta = \frac{N}{L_x L_y}\frac{\Delta}{r}F_{\mathrm{LJ}}$$

$$\begin{aligned}
F &= \int \mathrm{dsF_z(s)} = \int_{\mathrm{p_o}}^{\infty}\int_0^{2\pi} \mathrm{dpd}\phi\mathrm{pF_z(s)} = 2\pi\int_{\mathrm{p_o}}^{\infty}\mathrm{dppF_z(s)} = 2\pi\int_{\mathrm{p_o}}^{\infty}\mathrm{dpp}\frac{\mathrm{N}}{\mathrm{L_xL_y}}\frac{\Delta}{\mathrm{r}}\mathrm{F_{LJ}} \\
&= \frac{2\pi N\Delta}{L_x L_y}\int_{p_o}^{\infty}\mathrm{dpp}\frac{1}{\mathrm{r}}\mathrm{F_{LJ}} = \frac{2\pi\mathrm{N}\Delta}{\mathrm{L_xL_y}}\int_{\mathrm{p_o}}^{\infty}\mathrm{dpp}\frac{1}{\left(\mathrm{p}^2+\Delta^2\right)^{1/2}}4\epsilon\left(\frac{6\sigma^6}{\mathrm{r}^7}-\frac{12\sigma^{12}}{\mathrm{r}^{13}}\right) \\
&= \frac{8\pi\epsilon N\Delta}{L_x L_y}\int_{p_o}^{\infty}\mathrm{dp}\frac{\mathrm{p}}{\left(\mathrm{p}^2+\Delta^2\right)^{1/2}}\left(\frac{6\sigma^6}{\left(\mathrm{p}^2+\Delta^2\right)^{7/2}}-\frac{12\sigma^{12}}{\left(\mathrm{p}^2+\Delta^2\right)^{13/2}}\right) \\
&= \frac{8\pi\epsilon N\Delta}{L_x L_y}\int_{p_o}^{\infty}\mathrm{dpp}\left(\frac{6\sigma^6}{\left(\mathrm{p}^2+\Delta^2\right)^{4}}-\frac{12\sigma^{12}}{\left(\mathrm{p}^2+\Delta^2\right)^{7}}\right) \\
&= \frac{8\pi\epsilon N\Delta}{L_x L_y}\left(\frac{6\sigma^6}{6\left(p_o^2+\Delta^2\right)^3}-\frac{6\sigma^{12}}{6\left(p_o^2+\Delta^2\right)^6}\right) \\
&= \frac{2\pi N\Delta}{L_x L_y}4\epsilon\left(\frac{\sigma^6}{\left(p_o^2+\Delta^2\right)^3}-\frac{\sigma^{12}}{\left(p_o^2+\Delta^2\right)^6}\right) \\
&= \frac{2\pi N}{L_x L_y}\begin{cases} 4\epsilon\left(\frac{\sigma^6}{\Delta^5}-\frac{\sigma^{12}}{\Delta^{11}}\right) & |\Delta| > R_c \\ 4\epsilon\Delta\left(\frac{\sigma^6}{p_o^6}-\frac{\sigma^{12}}{p_o^{12}}\right) & |\Delta| \le R_c \end{cases}
\end{aligned} \tag{4.20}$$

$$p_o = max\{\sqrt{R_c^2-\Delta^2}, 0\}$$

For $|\Delta| > R_c$ the term inside the brackets is exactly the same as the solution in the previous section.

5. Parallel Programming

Traditionally software has been written for serial computation, where a problem is divided in discrete series of instruction sets. Instructions are executed sequentially on a single processor, with only one instruction being executed at any point in time as it is schematically shown in Fig. 5.1.

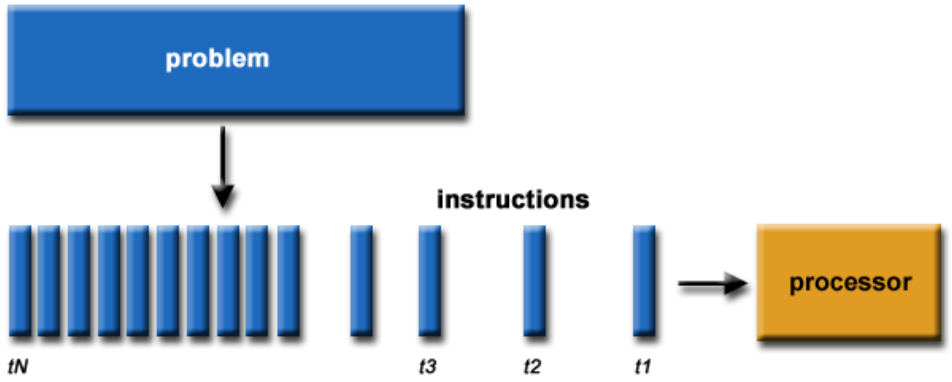

Figure 5.1: Example of serial computing [16]

The major focus of the industry for improvement of the processing power from the mid 80s was the increase of the density in transistors fitted in integrated circuits and the increase of the operating frequencies. In time the advancements in those fields increased so much as to reach the physical barriers. While the transistor density still increased, frequency scaling had reached power consumption and cooling barriers. If we define the capacitance being switched per clock cycle as C, the operating voltage V and the processor's frequency F, then the power consumption P of a processor is: $P = CV^2F$.

As we can see, with the increase of the operating frequency, there is increase of the power consumption and consequently of heat production. In addition the increased transistor density makes heat dissipation more difficult and thus the cooling techniques required to counter the

heat production become expensive and more complex. This brought a halt on the efforts for higher processor frequency in 2004 when Intel canceled its then current processor lineup for a focus on multi-core processors [17]. With the end of frequency scaling, the transistors which are no longer needed to facilitate frequency scaling were used to add extra hardware, such as additional cores to share the total workload.

Interest in parallel computing had appeared as back as the early 60s with the creation of the first supercomputers namely IBM's STRETCH and the LARC that solved problems of interest of the scientific community.

In layman's terms parallel computing is the simultaneous use of multiple compute units to solve a computational problem. The problem is initially broken into discrete parts that can be solved concurrently. Then we proceed to break each part in a series of instructions. Different instructions are processed simultaneously on different processors. A scheme summarizing these steps is shown in Fig. 5.2.

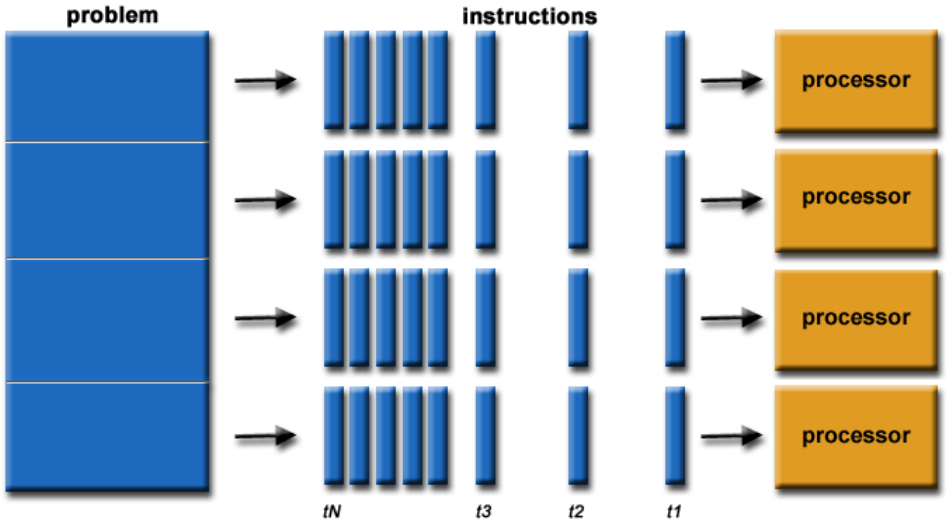

Figure 5.2: Example of parallel computing [16]

Therefore any computer consisting of more than two physical cores is capable of parallel computing. We can increase the scale by using computers that support multiple processors or in addition connecting multiple computers via network which constitutes what we refer to as a cluster.

At this point we should note some limitations of parallel computing. The challenge of parallel computing is that not all problems are easily parallelizable, and some of them are not parallelizable at all. In a serial

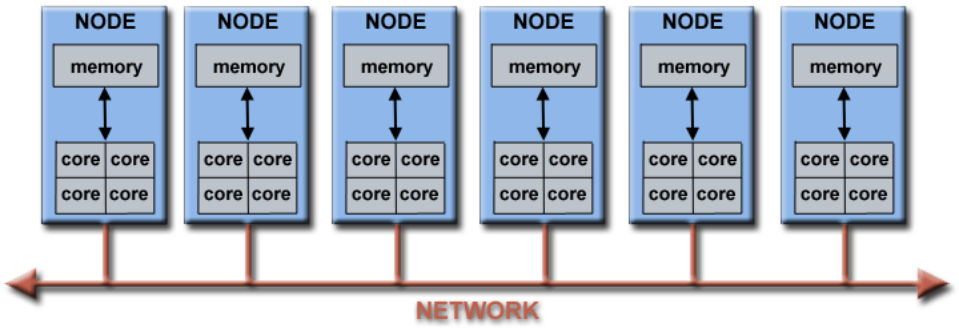

Figure 5.3: Example of a cluster computer [16]

algorithm one operation is performed after another and at the time it is performed its input data are assumed to be up to date. In a parallel algorithm one has to decompose the problem in such a way that parts of it can be processed, without the need for data that depends on other parts of the problem processed elsewhere. Different parts of the problem are then run in parallel and data are exchanged between those parts only when needed. Some issues arise during such a decomposition. No part of the processing should remain idle waiting for data from other parts, at least not for long. The amount of exchanged data should be minimal, as data transfer between processing units comes at a cost. Imbalances and bad synchronization make the whole set of processing units wait for the slowest part, so any effect of bad parallelization is essentially scaled by the number of processing units. Even if the problem can be made parallel to a great amount, writing the parallel software for solving it, is not a trivial thing in the sense that a large number of optimizations must be made for the overall performance to be acceptable. These optimizations may largely depend on the architecture of the used computing system and thus may not be easy to be generalized. Erroneous results are much more difficult to trace as data and processing are delocalized, and the problem decomposition usually adds significant complexity to existing serial algorithms. Finite communication speeds between parts of the problem almost always limit the amount of parallelization that can be achieved. What is the worst of all, is that there are often algorithmic barriers that limit the potential degree of parallelization of a problem. Nevertheless, much work has been done in parallel computing, and near optimal solutions exist for a great range of algorithmic problems. It is usual to know beforehand, what is the optimal performance one should expect to achieve, when handling a problem of a certain class.

In parallel computing, using only CPUs, there are two main models with their main distinction on how they distribute the data to be operated on.

Distributed memory model

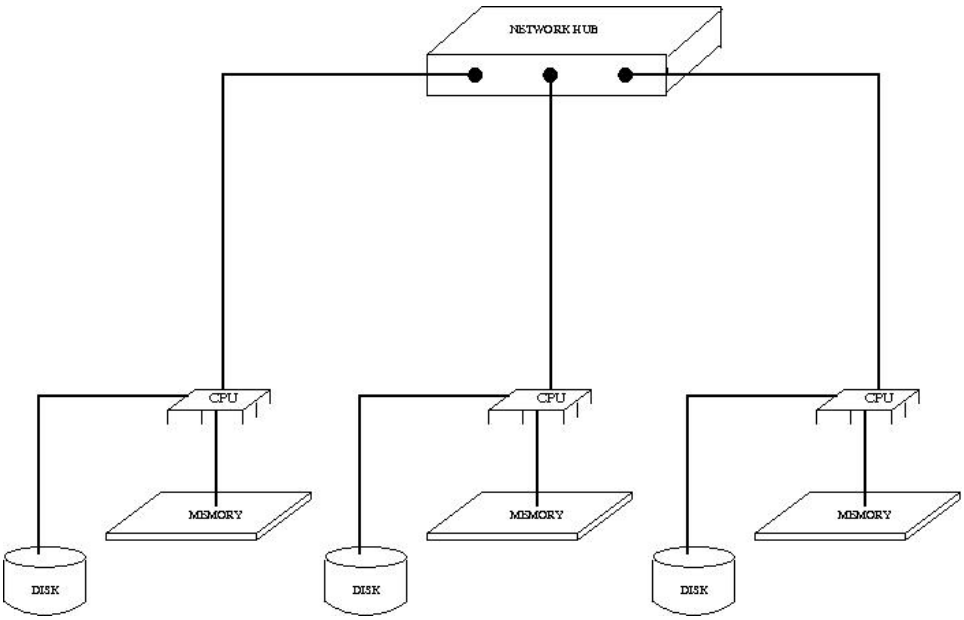

Figure 5.4: Example of a distributed memory model

The distributed memory model was the most common in the past due to computers being single-core with their own memory. In the distributed memory model each processing unit operates on data that is stored in its private memory and cannot be accessed by other processing units. Any data exchange between processing units is explicit, and some form of communication has to take place between processing units in order for it to happen.

For example in Fig. 5.4 we see three single processor computers interconnected by a network each with its own data in its private memory. Any data that needs to be exchanged has to be transfered through the network. Whenever remote data are needed, they must be explicitly requested by the algorithm to be transmitted or received, therefore a common problem is how to distribute the data and the cost of the communications necessary for the distribution.

A widespread programming library specification for message-passing used to code in the distributed memory model is MPI (Message Pass-

ing Interface) which is commonly used and is proposed as a standard by a broad selection of vendors and implementors. It is an add-on library available for multiple programming languages including but not restricted to C, C++, Fortran and a wide spectrum of operating systems like Linux, Windows and Mac OS X.

Shared memory model

In the shared memory model all the data that is being operated on is implicitly shared between all the processing units (see Fig. 5.5). There is no need for the algorithm to communicate data between the different processors, but care must be taken to synchronize updates and access to common data. While there are not explicit communication costs in the shared memory model, there are costs involved in the sharing of the data. Updates to a single datum shared from multiple processing units must still be made in a serial way and access to that datum must usually be somehow synchronized. The hardware taking care of the uniform appearance of memory to different processing units, must make sure that updates to a memory location from a processing unit are visible to other processing units as well. There are cases when this might lead to excessive processor-to-memory communications that cripple parallel performance.

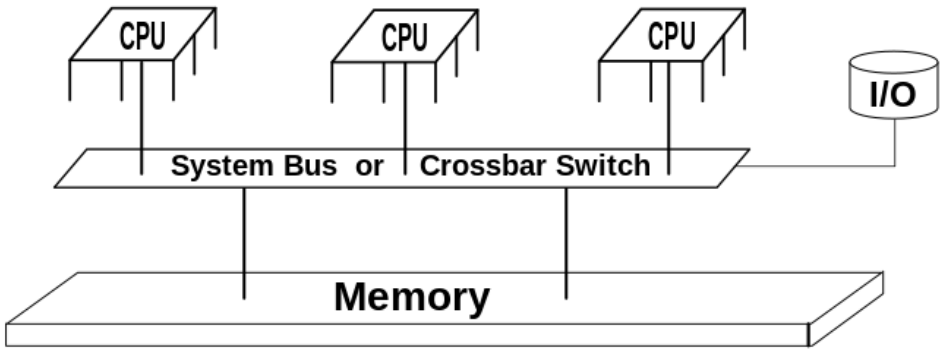

Figure 5.5: Example of a shared memory model

Generally shared memory parallelism is offering much higher performances than distributed memory parallelism, but when performance is not as high as expected it is more difficult to understand how it can be improved, as it is not always obvious how the complex hardware and software that implement shared memory operate. A widespread

and portable compiler extension exists that supports shared memory programming called openMP and has been around for many years making it quite mature. It is available for multiple programming languages including C, C++ and Fortran, and for multiple operating systems, including Linux, Mac OS X and Windows. It is primarily focused on the parallelization of loops that operate on sets of data, but it also provides directives for breaking up the parallel work into sections and sets of tasks. It is designed so that existing serial code, or at least big parts of it, can be easily parallelized, by minimal code changes that explicitly state to the compiler which data are shared between processes and how this sharing should be done.

Computing clusters much like the one used to run our simulations are composed of nodes with multiple processes and adequate memory interconnected with high speed networking. The cluster we work with has 13 nodes. Nodes 1-8 have two 6-core processors with 16 gigabytes of memory while nodes 9-13 have two 12-core processors with 32 gigabytes of memory, totaling 216 cores with 288 gigabytes of memory. To achieve the optimum performance of such a system we have to use the shared memory approach internally on each of the nodes and the distributed memory approach to combine the resources from all the nodes. In other words we use openmp for the parallelization in each node and MPI for the inter-node communication.

Before we describe openMP and MPI in the next two sections, we will briefly describe how to measure the performance of a parallel algorithm using the speedup function derived from Amdahl's law [18]:

$$S(P) = \frac{T_1(S)}{T_N(P)} \tag{5.1}$$

- S is the speedup factor
- $T_1(S)$ is the time required for the problem to be run serially
- $T_N(P)$ is the time required for the problem to be run parallelized by N threads

5.1 OMP

OpenMP is not just a library but also a specification for a set of compiler directives and environment variables. OpenMP works with threads. Threads are concurrently executing blocks of code which share access to a common memory space. The program is run once but it spawns multiple threads that share the workload of the computations. In general the operating system will create a number of threads equal to the number of cores (M number of cores) that the processor has, so its full potential is used. Also it is generally meaningless to create more than M threads as some of them would have to share a single processor's time.

Threads have been used to share a single processors time making what is known as multitasking available long before multi-core processors. Multitasking is when a single processor or even a single core runs many programs at the same time, by slicing them up and running small parts of them interchangeably, one after another, so that is seems to the user they are all running at the same time, while in reality they are being executed concurrently. That happens because threads that are not actively using much processing time but are waiting on some condition before they take action consume minuscule processing time, so the system may be essentially in an idle state, even though many threads may be running. On a modern multi-core processor multi-threading is the natural way of exploiting its potential where instructions from the same program are executed simultaneously on multiple cores.

The process of starting, synchronizing and terminating threads for parallel code can be done in multiple ways using different programming languages, but it is a quite tedious task, as different kinds of threads must be created and especially due to the managing of the synchronization. There is also the added problem of portability, as one wishes to be able to port a parallel program to a different operating system or compiler without having to modify the source code. OpenMP has been created to provide a straightforward, consistent and portable way to implement parallel shared memory algorithms, by making use of the multi-threading capabilities of modern computers. It provides constructs that inform the compiler how to automatically start and manage thread groups that share the load of processing.

```
#include <omp.h>
#include <iostream>
#include <math.h>
#include <chrono>
using namespace std;

int main() {
  // Random mathematical sum in serial code
  auto start_serial = std::chrono::high_resolution_clock::
    now();

  double serial_sum=0;
  for (int i=0; i<100000000; i++)
  serial_sum += pow(i,10);

  auto finish_serial = std::chrono::high_resolution_clock::
    now();


  // Same random mathematical sum in parallel code
  auto start_parallel = std::chrono::high_resolution_clock::
    now();

  double parallel_sum = 0;
  #pragma omp parallel for default(none) reduction(+ :
    parallel_sum) num_threads(4)
  for (int i=0; i<100000000; i++)
  parallel_sum += pow(i,10);

  auto finish_parallel = std::chrono::high_resolution_clock
    ::now();

  // Output of elapsed time
  std::chrono::duration<double> elapsed_serial =
    finish_serial - start_serial;
  std::chrono::duration<double> elapsed_parallel =
    finish_parallel - start_parallel;
  std::cout << "Elapsed time serial: " << elapsed_serial.
    count() << ", elapsed time parallel: " <<
    elapsed_parallel.count() << endl;
}
```

Listing 5.1: openMP example

Openmp in C and C++ works primarily with compiler pragma omp directives. The “pragma omp parallel” directive informs the compiler that the following block of code is to be run by simultaneous threads followed by extra arguments that can specify multiple things. One of the most common ones is that the compiler should not make any as-

number of threads	time [s]
1	6.2857
2	3.2114
4	1.7099
6	1.2422
8	0.9639

Table 5.1: Time required for code in serial and in parallel with the use of openMP.

sumptions about which variable is local to a thread and which is shared by all threads, declared with the "private()" and "shared()" clause respectively. In our example "pragma omp for" instructs the compiler that it should not run the block of code just using multiple threads but that the loop's iterations should be broken down per thread - so each thread takes care of 1/Nth of the loop's iterations. The reduction clause is considered a private type of variable and in our case "parallel_sum" is used to sum the values from all of the threads very own private "parallel_sum" into the "parallel_sum" variable that we have declared outside the parallel region. We should note that the "pragma omp parallel for" directive combines two openMP directives into one, something that is not always the correct thing to do as a series of successive loops may need to be executed by multiple threads. Finally "num_threads()" explicitly sets the number of threads that should be used.

To demonstrate how easily a serial code can be parallelized and subsequently sped up we used an example openMP code - see [5.1] - in serial (1 thread) and in parallel (2,4,6,8 threads) and we provide the results in Tab. 5.1. The system it was run on was a node with dual CPU E5-2680 v3 and 32 gigabytes of memory.

In Fig. 5.6 we can see Tab. 5.1 plotted and it is evident that a) the time needed for the example run is decreasing as the number of threads increases and b) the decrease in time is not constant as the threads increase, so there is a performance loss. We can take a better look at the performance by looking at Fig. 5.7 which is a representation of the speedup. The continuous line shows how the performance would be if the parallelization scaled perfectly with the increase of threads in a 1:1 ratio, with absolutely no loss, therefore a theoretical take. The rectangle points paint us the real image. Even for a simple loop as the one we used, albeit with absolutely no optimization, the scaling is far

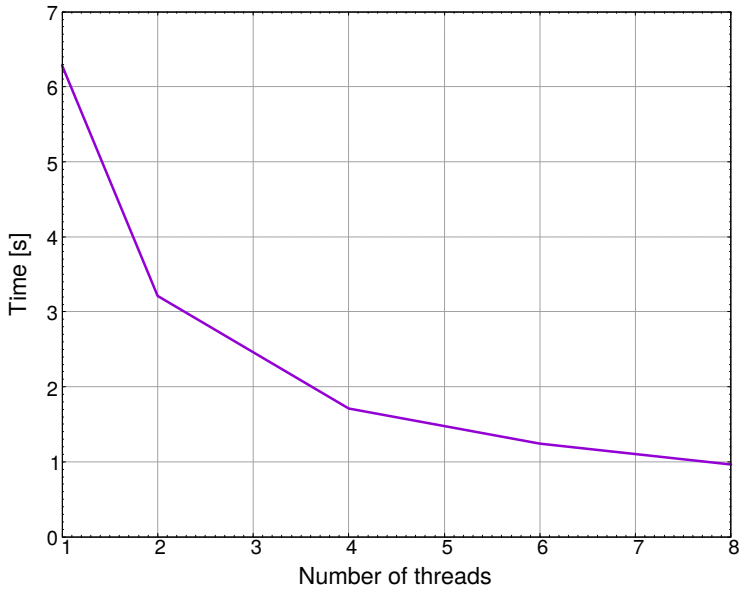

Figure 5.6: Plot of time needed for the run, as the number of used threads increases.

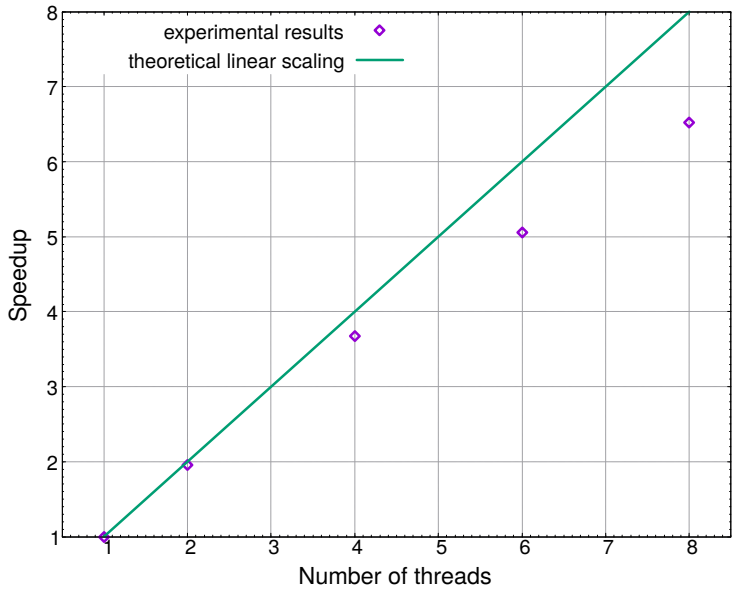

Figure 5.7: Speedup representation.

from nominal especially for the 8 and 6 threads runs. Still such a result is to be considered as a problem well parallelized and portraits the clear benefits of parallelization.

5.2 MPI

As we have mentioned before MPI does not add to the language syntax but is a library of routines. Multiple instances of a program run at the same time and the MPI library is used to handle communication and synchronization between those instances. Groups of instances (specific parts of the code) of a program may collectively create communicators. Any communication or collective operation must take place in the context of a communicator. There can be no data exchange without the use of a communicator. The global "MPI_COMM_WORLD" communicator is the default communicator created automatically by MPI, so in the simplest scenario only the global communicator is used during the whole execution of a program. When it's required by a set of processes to differentiate their execution path they may create their own distinct "communication space" by the derivation of sub-communicators from the global communicator. An instance of a program is not limited to one communicator but can take part in many different communicators at the same time. Each instance of the program is called a "rank" in the context of a communicator. Each rank has a unique, sequential number assigned to it and so if a program runs on five processors the ranks of its instances will be 0,1,2,3 and 4. When an instance of the program participates in multiple communicators it may be assigned a different rank number at each of the communicators. The ranks numbers are used to identify program instances in communicators. It is the means by which an instance of the program identifies itself, and the means by which the MPI library is instructed, through the code, with which other instance of the program it should perform an operation with.

Data are exchanged between ranks in the form of messages. When one rank sends a message another must receive it. Messages have a sender, a receiver, a type, and carry some data. Messages of different types are distinguished between them by an integer tag. A sender may send many messages of different types to a receiver. The receiver may choose which message types he wishes to receive at a time from that specific sender. If the sender, receiver and message types do not match at both ends of a send/receive operation the data exchange fails.

Sends and receives can be processed synchronously or asynchronously. When sending data one may choose to wait until a matching receive has started before continuing, or choose to continue doing other stuff till the send operation successfully completes. Likewise on the receiving end one can choose to wait until a message he expects is received before he continues, or may just check if such a message exists and

do other stuff until the wanted message arrives. Apart from point to point send-receive operations, MPI also supports some sorts of collective operations. The simplest of them is the reduction operation, where sums, products, minimums etc of values residing at different ranks are calculated and made available to these ranks. Others are the broadcast operation where some data residing in one rank are made available to all other ranks, or the scatter operation where different parts of data originating from a single rank are made available to all other ranks.

6. MDPAR Algorithm

6.1 Description of MDPAR

MDPAR is a parallel molecular dynamics simulator implemented in C++ using MPI and openMP. The parallelization is done mainly by domain decomposition by splitting the simulation volume to smaller (equal) parts and assigning one part to each processor involved. The rank (thread) number is easily mapped to the rank coordinates (i,j,k) and so we acquire the neighbor rank numbers at each direction.

1. When the simulation starts, the configuration is read from disk by the root rank (serial process). Global configuration data are transmitted to the other ranks using the MPI broadcast function.

2. The dimensions of the decomposition are decided accordingly to the number of processes available. Using MPI's scatter, atoms and their bonds are sent to their respective ranks by using a temporary linked list to classify atoms to ranks.

3. The coordinates of atoms assigned to neighboring ranks are needed. We limit it to the immediate neighbors by demanding that the potential cutoff distance does not exceed the dimension of the volume assigned to each rank. As such at most there can be 26 neighboring ranks.

4. The communications required by the ranks on step 3 are split in two parts. The first stage is when the neighbor lists of the atoms are being constructed – to determine the atoms that are close enough to interact. Atoms that may interact with atoms of a neighbor are considered the ones whose distance from the boundaries is not greater than the potential cutoff distance.

5. The check for cross rank interaction happens by having each rank send to its neighbors the coordinates of atoms it holds. The re-

mote rank replies with which atoms actually have cross rank interactions.

6. Steps 4 and 5 are sped up by using the linked cell list described in step 2.

7. For part 2, for each integration step the coordinates of atoms needed by remote ranks must be transmitted to them. Furthermore some reaction forces will be applied to the transmitted atoms by the receiving ranks which will have to be gathered and summed up to the total force acting on the local atoms.

8. After some integrations (order of 10) the neighbor lists have to be reconstructed. A check is done to verify if some atom's coordinates are out of bounds and to identify to which neighbor rank they must be transferred to.

Classes / methods of importance

- Loading of initial configuration – SimulationSetup class and specifically the initialize method.
- Instantiation of a valid simulation – startSimulation method.
- Communications – mainly by the Simulation class.
- Identification of neighbor interactions and preparation for integration – makeLists method.
- Relocation of atoms amongst ranks – transferAtoms.
- Transfer of coordinates – checkGetNeiPositions.
- Transfer of reaction forces – getf and get_fref methods.

6.2 Implementation of Tail Correction

6.2.1 Input Configuration

- From the LJ case we conclude that either δ_{layer} or N_b should be given as input.
- Also a boolean value should be read to indicate whether inhomogeneous tail correction should be applied or not.

- These parameters should be provided in the *.cfg* file which is read by SimulationSetup.
- The parameter names and help strings should be described in SimulationSetup::initKnownArgs()
- They should be read in member variables of SimulationSetup in SimulationSetup::initialize
- They should be broadcasted to mpi nodes in SimulationSetup::startSimulation
- They should then be assigned to each simulation node in SimulationSetup::startSimulation using some helper function like Simulation::setDeltaNbLayer(thickness) and Simulation::setNbTail(on/off)

6.2.2 Calculation of N(J layer)

- Should be performed after the linked cell list creation, inside makeLists, every so many calls, as the density profile is not expected to vary rapidly
- A container function putAtomsInLayers() has been created and waits implementation
- The density histogram should be mostly implemented in a different class to avoid core bugs and a pointer to it should be kept in the Simulation class
- Domain and mpi info can be retrieved through Simulation::m_domain

7. Results

In the following we give results from our simulations.

7.1 Comparison of results obtained by different codes and with / without tail correction

We begin by comparing the results obtained by using home-made MDPAR code with those obtained by commercially available GROMACS package. The property that we compare with is the density profile which is the amount of mass as a function of distance r from the graphene for S1 while for F1,F2 is the distribution of mass of the polymer film. Density is a very important property and it is widely known to affect all system properties. We define as reference system the S1 system described in 3.1.1 and simulated with GROMACS. The advantage of using this particular system is that it has been previously simulated by members of our group [19]. We run S1 on MDPAR without tail correction, with R_{cut}=1.0 for 2ns. The same system is then simulated by GROMACS and their respective density histograms are compared. In Fig. 7.1 we can see the characteristic peak near the graphene layer, typical for polymer absorption on the surface where the attractive forces between the graphene and the polyethylene are quite strong. We can also observe that at distances greater than approximately 2nm there is a plateau in the density function, which value corresponds to the density in the bulk-like region. At distances $\approx$ 18nm the function decays to 0, as the polymer creates a free surface in contact with vacuum. The interactions between the frozen graphene layer and the polyethylene monomers causes a movement of the bulk towards the bottom of the simulations box evident by the shift of the tail towards the beginning of the axis.

In Fig. 7.2 we present the same data in Fig. 7.1 but we also include

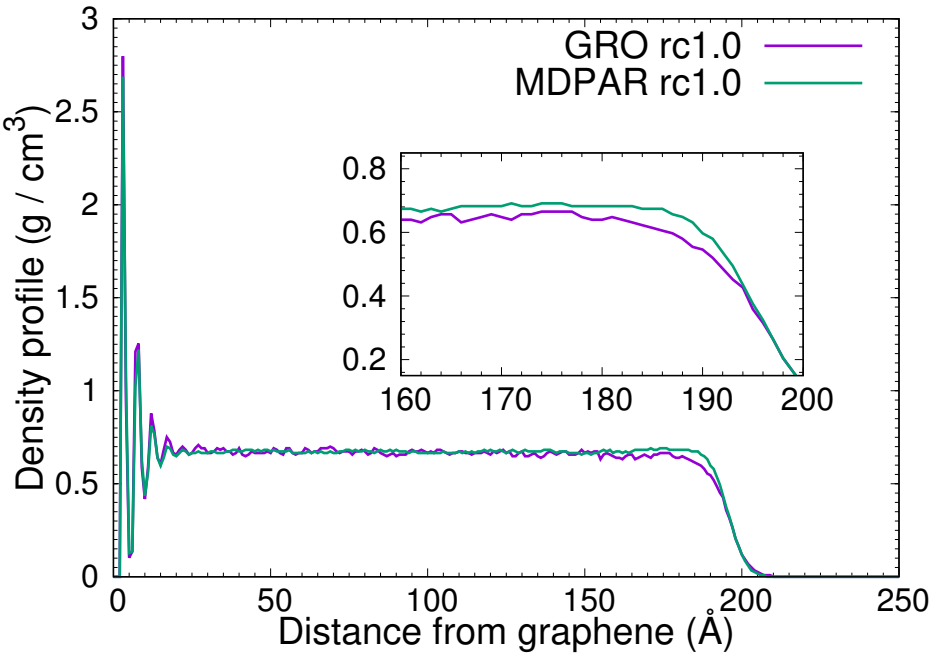

Figure 7.1: Gromacs and MDPAR density histogram comparison for R_{cut}=1.0

the density histogram of a S1 system that is simulated for 2ns via MDPAR for R_{cut}=1.0 and with tail correction. We can detect a tendency of the bulk-region density to be systematically higher than in the case of systems with no tail corrections (compare light blue line with either green or purple line in Fig. 7.2) as well as a shift of polymer / vacuum inteface. The bigger forces derived by the addition of energy due to the tail correction potential lead to higher density and a small decrease of film thickness.

To evaluate the performance of the Tail Correction we will compare the density histogram of the Tail Correction run with the following two runs:

- Gromacs run for 2 ns with R_{cut}=1.5
- MDPAR run for 2 ns with R_{cut}=1.5, without Tail Correction

We chose $R_{cut} = 1.5$ as this is widely considered as enough distance to include the correction by tail correction, simulated by both packages to further show the similar results.

As we can see in 7.3 the curve corresponding to tail correction appears to have larger bulk density. The behavior of the "tail" supports

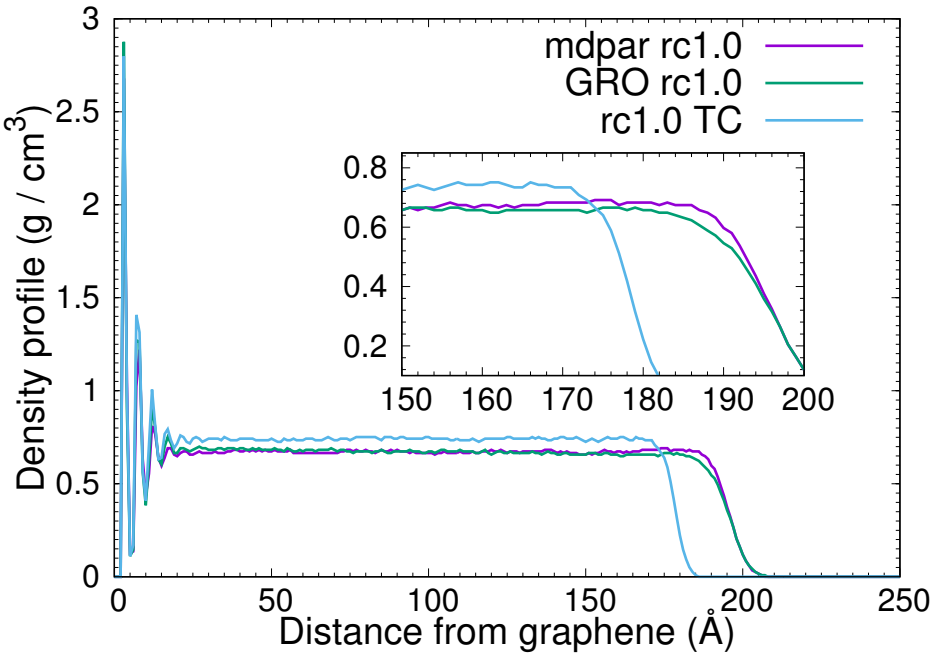

Figure 7.2: Gromacs, MDPAR density histogram comparison for R_{cut}=1.0, MDPAR density histogram comparison for R_{cut}=1.0 with Tail Correction

that statement since the polyethylene bulk has shrunk due to the increased forces.

In order to check the possible effect of the frozen graphene layer on the tail correction implementation we remove the frozen graphene layer from our system and repeat the process. The polyethylene film is now surrounded in both sides by vacuum. Due to the absence of the Lennard-Jones potential caused by the interaction of the polyethylene and the graphene we expect not to see the characteristic peak. The density profile should have a symmetry to it as there are no external forces in the vacuum and so the polyethylene acts like a sphere in the vacuum.

Our new S2 system's properties can be seen in 3.1.2.

In Figure 7.4 we confirm our expectations regarding the polyethylenes behavior in vacuum. It is also clear that there is no effect of the frozen graphene layer on our implementation.

As the movement of the bulk is not easy to discern, we will try to make it more easily understood by plotting a density graph Fig. 7.5 containing the following measurements: GROMACS using an R_{cut}=1.0,1.5

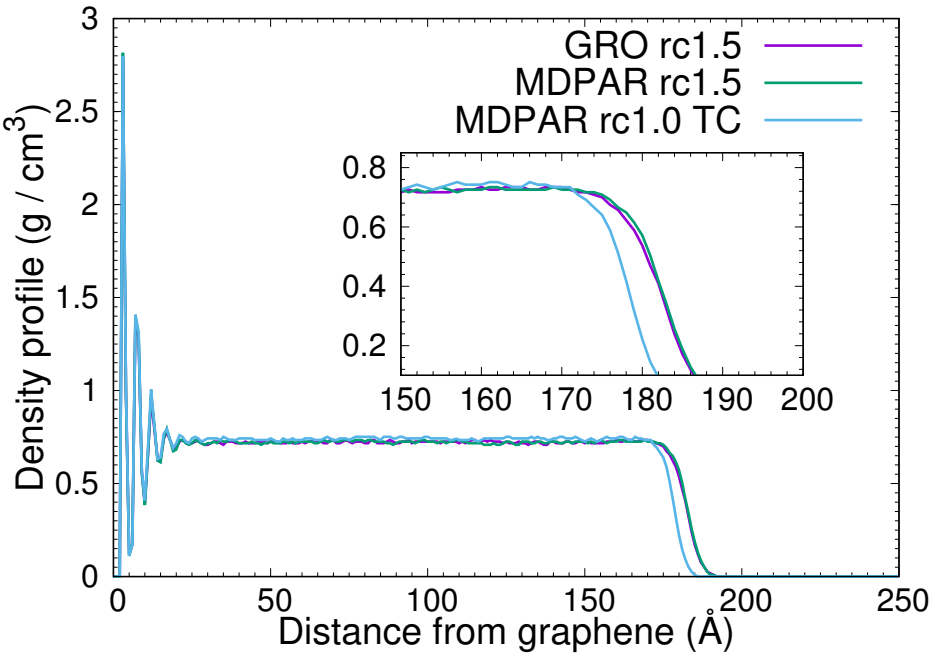

Figure 7.3: Gromacs, MDPAR density histogram comparison for R_{cut}=1.5, MDPAR density histogram comparison for R_{cut}=1.0 with Tail Correction

without the use of tail correction and MDPAR using R_{cut}=1.0 with the use of both approaches for tail correction. In the above figure it should be easier to discern the progressive differences beginning from R_{cut}=1.0, followed by R_{cut}=1.5 and finally the tail correction implementation. As we can see the implementation of tail correction exceeds the R_{cut}=1.5 a value that is generally deemed adequate to cover the energy loss from the tail correction. This potential overestimation of the tail correction is not something we should really worry about as there is no reason that a R_{cut}=1.5 is great enough to contain all the energy corrections. Therefore, it creates the question for which R_{cut} we will get approximately the same value as with tail correction and that is something we will investigate in the next section.

7.2 Convergence of the Tail Correction

As we have mentioned before the total energy is equal to the Lennard-Jones potential in combination with the tail correction component, if present (see eq. 4.1, 4.2). While performing simulations one could won-

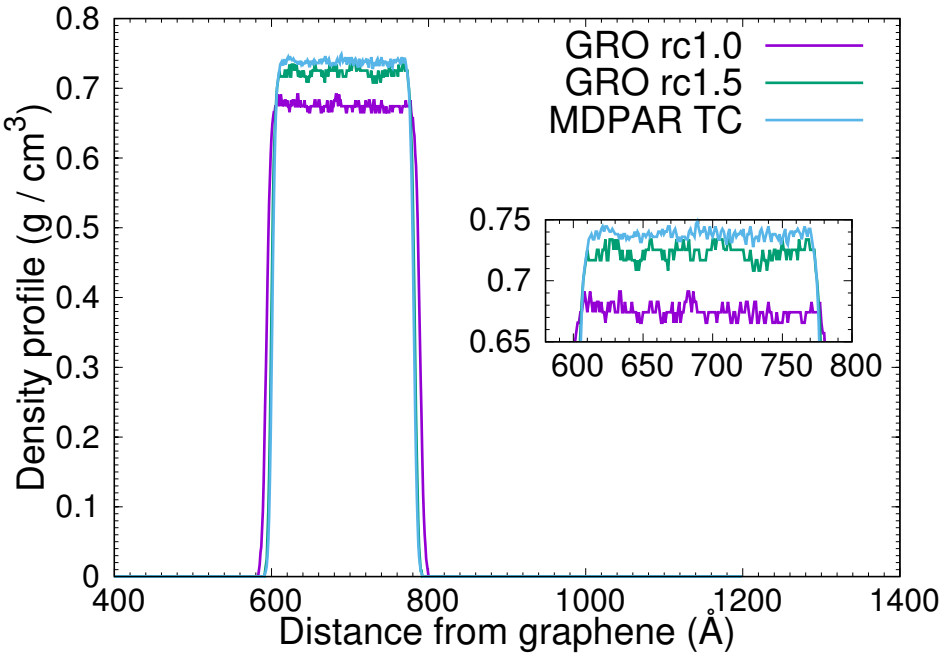

Figure 7.4: Polyethelyne in vacuum. Density histogram comparison for R_{cut}=1.0, R_{cut}=1.5 with Gromacs and MDPAR R_{cut}=1.0 with Tail Correction

der what would be the required value of R_{cut} in a simulation with no tail correction in order to achieve the same total energy as in a simulation using tail correction. In other words, what is the R_{cut} distance for which the potential converges to the energy with the tail correction implementation.

To answer the above dilemma we created the following setup. We used the F2 system with tail correction and $R_{cut} = 1.0$ to run a simulation and collected the total energy values ($V_{LJ} + V_{TC}$). Next we run a set of simulations, this time without the use of tail correction and with R_{cut} varying from 0.9 nm up to 4.9 nm. The results are presented in Tab. 7.1 where ΔV is equal to the total potential energy from the simulation with tail correction minus the V_{LJ} from a simulation without tail correction.

The total potential energy in our simulations with tail correction was equal to -57165.098 [Kcal/mol]. As we can see from Table 7.1 for a given error of approximately 0.1%, the computation of the LJ converges only for values of $r_{\text{cut}} \approx 3.6$nm. Such values are not feasible in MD simulations due to the extreme computational resources required as we

R_{cut} [nm]	V_{LJ} [Kcal/mol]	ΔV [Kcal/mol]	ΔV %
0.9	-49657.322	7507.775	84.881
1.0	-51725.525	5439.573	89.484
1.1	-53130.275	5439.573	92.406
1.2	-54084.306	3080.791	94.304
1.4	-55248.911	1916.187	95.532
1.6	-55907.327	1257.771	97.750
1.8	-56300.188	864.909	98.464
2.0	-56550.792	614.305	98.914
2.4	-56832.168	332.929	99.414
2.8	-56968.915	196.183	99.656
3.2	-57048.745	116.353	99.796
3.6	-57096.167	68.931	99.879
4.0	-57128.415	36.683	99.936
4.4	-57147.952	17.145	99.970
4.8	-57161.343	3.754	99.993
4.9	**-57163.981**	**1.117**	**99.998**
5.0	-57166.348	-1.251	100.002

Table 7.1: Convergence to TC

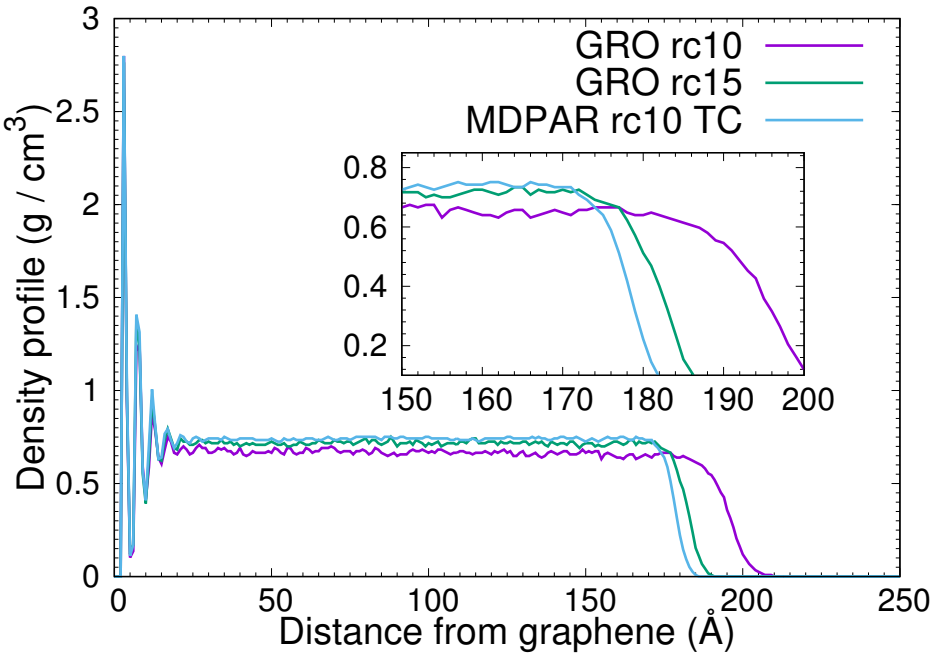

Figure 7.5: MDPAR with tail correction and GROMACS using R_{cut}=1.0,1.5 without tail correction

demonstrated earlier by comparing the time required for $r_{\text{cut}} = 1.0$ with 1.5 and 2.0. Clearly it shows that the assumption that using an r_{cut} of up to 1.5 to incorporate the potential energy from the tail correction is false and even in that case there is a noticeable systematic error introduced.

The contents of Table 7.1 have also been plotted and their representation can be seen in Fig. 7.6. The horizontal line represents the total potential energy of the simulation using tail correction.

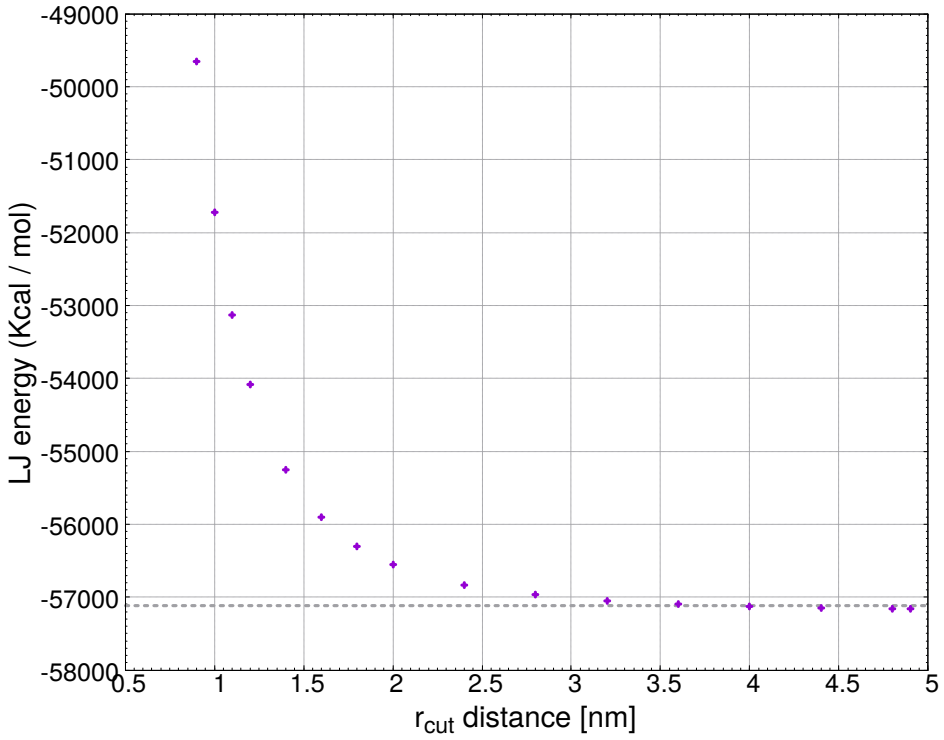

Figure 7.6: Plot of the Lennard-Jones energy as a function of the R_{cut} distance. The dashed line represents the total potential energy of the simulation using TC.

8. Conclusions

Here we have studied molecular systems through molecular dynamics simulations, as an example of a N-body problem. We have developed a new method to incorporate the tail correction potential energy in our calculations for an inhomogeneous system. Furthermore, we have shown the importance of precise tail correction calculations and how otherwise they introduce a notable systematic error affecting the system's properties. We proceeded to formulate an alternative derivation of the tail correction of the inhomogeneous system done by direct integration and have shown it arrives in the exact same equations. We implemented our tail correction methods in the MDPAR open-source package using MPI and openMP for parallelization of our code. We simulated using our implementations in two types of systems, polyethylene with a deposited frozen graphene layer and polyethylene surrounded on both sides with vacuum and compared the results using density histograms. Our results have shown clear differences with and without the use of our tail correction method and confirmed the importance of a proper calculation of the tail correction in inhomogeneous systems. In addition we have shown that usage of the truncated LJ potential with a large r_{cut} is clearly not enough even by using an r_{cut} as high as 1.5 which is widely regarded as the "upper limit" by the scientific community. We expanded by calculating the convergence of the tail correction potential energy and provided a r_{cut} value at which the systematic error is removed but at prohibitive computational cost. Future goals include further analysis of the model system, optimization and expansion of the parallel code used to implement our methods.

Bibliography

[1] J. L. Hennessy and D. A. Patterson, *Computer Architecture, Fifth Edition: A Quantitative Approach*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 5th ed., 2011.

[2] K. Johnston and V. Harmandaris, "Hierarchical simulations of hybrid polymer-solid materials," *Soft Matter*, vol. 9, pp. 6696–6710, 2013.

[3] B. Hess, C. Kutzner, D. van der Spoel, and E. Lindahl, "Gromacs 4: Algorithms for highly efficient, load-balanced, and scalable molecular simulation," *Journal of Chemical Theory and Computation*, vol. 4, no. 3, pp. 435–447, 2008.

[4] S. Plimpton, "Fast parallel algorithms for short-range molecular dynamics," *Journal of Computational Physics*, vol. 117, no. 1, pp. 1 – 19, 1995.

[5] J. Halverson, T. Brandes, O. Lenz, A. Arnold, S. Bevc, V. Starchenko, K. Kremer, T. Stuehn, and R. D., "Espresso++: A modern multiscale simulation package for soft matter systems," *Computer Physics Communications*, vol. 184, Apr. 2013.

[6] T. Chazirakis, "empty," *empty*, empty.

[7] D. Frenkel and B. Smit, *Understanding Molecular Simulation*. Orlando, FL, USA: Academic Press, Inc., 2nd ed., 2001.

[8] D. McQuarrie, *Statistical Mechanics*. University Science Books, 2000.

[9] H. C. Andersen, "Molecular dynamics simulations at constant pressure and/or temperature," *The Journal of Chemical Physics*, vol. 72, no. 4, pp. 2384–2393, 1980.

[10] W. G. Hoover, "Constant-pressure equations of motion," *Phys. Rev. A*, vol. 34, pp. 2499–2500, Sep 1986.

[11] S. Piana, K. Lindorff-Larsen, R. M. Dirks, J. K. Salmon, R. O. Dror, and D. E. Shaw, "Evaluating the effects of cutoffs and treatment of long-range electrostatics in protein folding simulations," *PLOS ONE*, vol. 7, pp. 1–6, 06 2012.

[12] D. Fritz, K. Koschke, V. A. Harmandaris, N. F. A. van der Vegt, and K. Kremer, "Multiscale modeling of soft matter: scaling of dynamics," *Phys. Chem. Chem. Phys.*, vol. 13, pp. 10412–10420, 2011.

[13] M. G. Martin and J. I. Siepmann, "Transferable potentials for phase equilibria. 1. united-atom description of n-alkanes," *The Journal of Physical Chemistry B*, vol. 102, no. 14, pp. 2569–2577, 1998.

[14] J.-P. Ryckaert and A. Bellemans, "Molecular dynamics of liquid alkanes," *Faraday Discuss. Chem. Soc.*, vol. 66, pp. 95–106, 1978.

[15] K. C. Daoulas, V. A. Harmandaris, and V. G. Mavrantzas, "Detailed atomistic simulation of a polymer melt/solid interface: Structure, density, and conformation of a thin film of polyethylene melt adsorbed on graphite," *Macromolecules*, vol. 38, no. 13, pp. 5780–5795, 2005.

[16] B. Barney, *Introduction to Parallel Computing*. Lawrence Livermore National Laboratory, 2007.

[17] L. Flynn, "Intel halts development of 2 new microprocessors," *New York Times*, 2004.

[18] G. M. Amdahl, "Validity of the single processor approach to achieving large scale computing capabilities," in *Proceedings of the April 18-20, 1967, Spring Joint Computer Conference*, AFIPS '67 (Spring), (New York, NY, USA), pp. 483–485, ACM, 1967.

[19] A. Rissanou and V. Harmandaris, "Dynamics of various polymer-graphene interfacial systems through atomistic molecular dynamics simulations," *Soft matter*, vol. 10, pp. 2876–88, 03 2014.