Chapter 1

Introduction: Water in Confined Environments

1.1 History and Motivation

Water has been a subject of scientific investigation for millenia[1,2]. In ancient times, for example, significant efforts in water research were motivated by the desire to obtain healthier, better tasting drinking water. People discovered that heating water might purify it, and they were also educated in cleansing effects of sand and gravel filtration and of boiling and straining processes[1]. In early medical research, Hippocrates and others investigated the "healing powers" of water used in both internal and external applications. In modern times, water research is an exceptionally diverse topic that includes not only health-related research but also, for example, investigations of water's role in the environment and in a wide range of chemical applications[2].

Water's behavior is strongly dependent on the structural environment in which it is confined, showing a set of unique characteristics in highly constrained environments[3]. For example, freezing water in the cracks of rocks at normal temperature speeds up weathering. In order to reveal these characteristics, experimental studies of water under confinement have been carried out since early in the 20th century[3]. When water is confined in different geometries[4-7], its anomalous behavior attracts much attention due to its relationship to systems in nature and within the human body, such as water in plants, fruits and blood vessels,

in foam, in micelles, on the surface and interior of proteins, in grooves of DNA, and so on. These cover several diverse fields of science, including biology, nanofluidic devices, inorganic materials and geology.

The origins of water's interesting and anomalous behavior in confined environments have been studied for several decades[8]. The equilibrium structure of confined water clusters is different from the structure of bulk water and ice under ambient pressure. This is due to the combined influence of the so-called packing effect and the overlapping of the potentials exerted by the walls.[8] Neutron diffraction experiments have also revealed the presence of cubic ice in confined environments, usually only observed in high pressure studies, at low temperature (-13℃) and ambient pressure[3,9,10]. In general, liquids next to a wall undergo stratification that extends 2-3 molecular diameters[11]. When the confined environment is narrow enough, the potentials exerted by the walls start to overlap. In this case, the fluid within the pore can be divided into two regions: organized fluid for the region close to the wall and confined fluids for the region in the inner core of the pore[12]. One result of the overlapping potentials is to densify the adsorbates, leading to a strong interaction among adsorbate molecules[13]. The packing effect can cause the imperfect packing for large molecules, such as carbon tetrachloride[14]. Hysteresis between adsorption and desorption isotherms can be observed in small pores as a result.

The influence a confined environment has on water behavior is a function of

the pore size, the interaction between water and the pore wall material, water-water interactions, and thermodynamic variables such as temperature and pressure.[15-20] In addition, hydrophobic forces have been found to significantly stabilize water structure in confined enviroments[18,19]. The hydrogen bonding networks of bulk water are modified when water enters a confined environment. The average number of hydrogen bonds per water molecule tends to decrease due to water-wall surface effects, while individual hydrogen bonds may become stronger. Part of the hydrogen bond energy that is lost in the confinement process is recovered through the interaction between water and the pore walls[21].

This thesis consists of a report of computational investigations of water and aqueous solutions in confined environments involving carbon nanotubes[22-24] (Chapter 3) and an idealized slit pore[25-27] systems (Chapter 4). The simulations were performed with a goal of providing information useful in for material design and future experiments on confined aqueous solutions.

1.2 Simulation

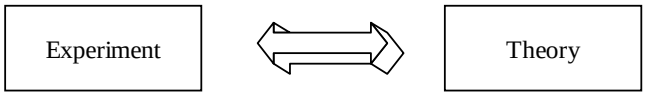
A diagram illustrating the traditional scientific research method involving an interplay between experiment and theory is shown in the top frame of Fig 1.1. In this method, data are collected from experiment, hypotheses or principles linked by logical or mathematical arguments are proposed, and following testing and refinement, explanatory theories are developed to explain areas of empirical reality and to predict results of future experiments. This procedure has been used since

the advent of scientific investigations and is still a productive avenue for research today.

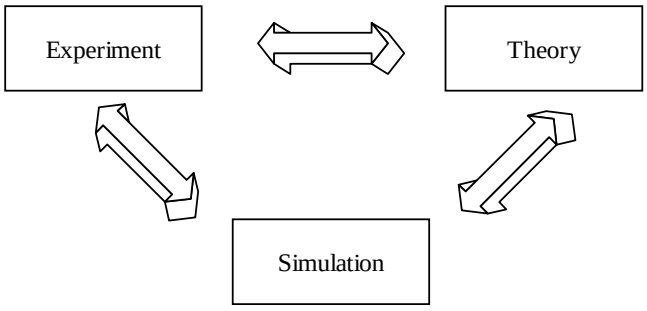
Following the development of electronic computers in the 1950s[28], an additional and powerful tool, computer simulation, has been added to this scientific method framework, as shown in the bottom frame of Fig 1.1. In this modified research method, computer simulations play several important roles. For example, a theory developed from experimental results may be validated for simplified model systems through computer simulations, providing inexpensive, preliminary feedback on the theory prior to full experimental evaluation. Simulations here play the role of an experiment to generate "exact" behavior of the model system[29]. In addition, computer simulations may be used to reveal information that is experimentally inaccessible, such as details of molecular structure or motion, that may aid in theory refinement. Finally, simulations may play a predictive role to guide the design of complicated experiments. As an example, suppose a molecular strategy for disease treatment is proposed in a long-term medical design project. Experimental evaluation of this strategy might require expensive and time consuming synthesis and testing of numerous drug candidates that might take decades of research to complete. By applying simulations tool to this project, drug candidates may be screened initially with respect to the molecular strategy, reducing the pool of potential candidates substantially and streamlining the subsequent experimental evaluation process. This will potentially increase the

design speed and save people's lives as soon as possible.

There are two popular approaches to performing molecular computer simulations, the Monte Carlo (MC) method and the molecular dynamics (MD) method. MD simulations involve integration of Newton's equations of motion to essentially create a time-dependent "experiment" of the system. The forces on the atoms are computed at each step and used to calculate the atom positions at the next step. Both thermodynamic and dynamical properties of the system can be obtained by the MD method. The MC method is an efficient alternative to MD suitable for use when only thermodynamic properties are needed. In this research, the MC method is used. Details of the method are introduced in chapter 2.



Traditional research method



Simulation Modified Method

Fig. 1.1 Relationship between experiment, theory and simulation.

In computer simulations, the statistical mechanics term “ensemble” is used to represent the thermodynamic constraints applied in a simulation. Specifically, an ensemble is an idealization consisting of a large number of mental copies of a system[30], each of which represents a possible state that the system might be in. Two ensembles, the grand-canonical ensemble (μ, V, T) and the canonical ensemble (N, V, T), are used in the simulations described here. In the canonical ensemble, the system is thought of as being in contact with a large heat reservoir. The system maintains a constant temperature by exchanging thermal energy with the reservoir. The heat capacity of the reservoir is large enough to maintain a fixed temperature (T). The particle number (N) and volume of the system (V) are also constant in the canonical ensemble. The grand-canonical ensemble is also in contact with a reservoir. However, it not only exchanges the energy but also exchanges particles with the reservoir to establish a constant chemical potential (μ) constraint. The grand-canonical ensemble is useful in adsorption studies because most of the time one would like to know the amount of material adsorbed as a function of the vapor pressure of the adsorbed material. Since vapor pressure is directly related to chemical potential, the grand canonical ensemble establishes the desired constraint[31]. Switching between the canonical and grand-canonical ensembles in the simulations is controlled by parameters that will be described in Chapter 5.

The process of performing molecular computer simulations can be broken down conceptually into several steps.

1. Defining the model. Interaction potential models defining how the components of a molecular system interact are the single most important set of input parameters for a simulation. The molecular models used in this work are standard ones, taken from various literature sources.
2. Initializing the system. All simulations begin with some starting configuration generated either “from scratch” or from the output of some previous simulation. Generating a suitable initial configuration for a simulation system is particularly important where equilibration is slow.
3. Equilibrating the system. Prior to data collection, it is important to allow a system to establish equilibrium at the chosen thermodynamic constraint values. Assessing when a system has reached equilibrium is sometimes difficult and very important.
4. Calculate equilibrium averages. Once a system is deemed to be at equilibrium, data collection and averaging of desired structural and thermodynamic properties may proceed.
5. Evaluation of results. Comparisons with experimental data or results from independent simulations are used frequently to validate simulation results.

1.3 Application

In this research, water and aqueous solution have been studied on various carbon-based materials by the Monte Carlo method. An outline of the material is

as follows. Chapter 2 introduces the general canonical Monte Carlo (MC) and grand-canonical Monte Carlo (GCMC) theories and describes the simulation program as it is applied to pure water. Optimization of the MC and GCMC codes is also described in this chapter along with several measurements used to validate the simulation code. Chapter 3 focuses on hydrated carbon nanotube (CNT) systems. Pure water with a single walled CNT is checked first with a goal of validating the simulation code. Various numbers of chloroform molecules are then inserted into the system, outside of the CNT. Chloroform structure and chloroform-CNT binding free energies are then measured as a function of chloroform content. Chapter 4 describes the behavior of water-methane mixtures adsorbed in idealized slit pores. Chapter 5 involves a description of the C++ simulation code including a discussion of the program structure and benefits obtained from using the C++ programming language. Finally, simulation source codes are collected in the Appendix along with code documentation that defines the roles of each subroutine and gives line-by-line descriptions of the code.

1.4 References

[1] Baker, M. N.; Taras, M. J. *The Quest for Pure Water – The History of the Twentieth Century*; volume I and II; AWWA: Denver, 1981.

[2] Outwater, A. *Water: A natural history*; Basic Books; New York, USA, 1996.

[3] Christenson, H. K. *J. Phys. : Condens. Matter. 2001, 13,* R95.

[4] Bhattacharyya, K.; Bagchi, B. *J. Phys. Chem. A 2000, 104,* 10603.

[5] Bellissent-Funnel, M. C. *Hydration Processes in Biology: Theoretical and Experimental Approaches*; IOS Press: Amsterdam, 1999.

[6] Fitter, J.; Lechner, R. E.; Dencher, N. A. *J. Phys. Chem. B* 1999, *103*, 8036.

[7] Fu, X.; Shen, W.; Yao, T. *Physical Chemical II*; Advanced Education of Press: China, 1996.

[8] Do, D.D.; Do, H.D. *Adsorption Science & Technology. 2003, 21*, 389.

[9] Steytler, D.C.; Dore, J.C.; Wright, C.J. *J. Phys. Chem. 1983, 87*, 2458.

[10] Steytler, D.C.; Dore, J.C. *Mol. Phys. 1985, 56*, 1001.

[11] Hansen, J.P.; McDonald, I.R. *Theory of Simple Liquids 2nd edition*. *1986* (London: Academic).

[12] Murata, K.; Kaneko, K. *Proceedings of the 4th IUPAC symposium on Characterisation of Porous Solids, Royal Society of Chemistry Special Publication. 1996*, No.213.

[13] Watanabe, A.; Iiyama, T.; Kaneko, K. *Chem. Phys. Lett. 1999, 305*, 71.

[14] Iiyama, T.; Nishikawa, K.; Suzuki, T.; Otowa, T.; Hijiriyama, M.; Nojima, Y.; Kaneko, K. *J. Phys. Chem. B 1997, 101*, 3037.

[15] Pregel, M. J.; Jullien, L.; Lehn, J. M. *Angew. Chem., Int. Ed. Engl.* 1992, *31*, 1637.

[16] Gelbart, W. M.; Ben-Shaul, A. *J. Phys. Chem.* 1996, *100*, 13169.

[17] Lum, K.; Chandler, D.; Weeks, J. D. *J. Phys. Chem. B*, 1999, *103*, 4560.

[18] Chandler, D. *Nature* 2005, *437*, 640.

[19] Lum, K.; Luzar, A. *Physical Review E* 1997, *56*, 6383.

[20] Zangi, R. *J. Phys: Condens. Matter. 2004, 16*, S5371.

[21] G. Hummer, J.C. Rasaihah & J.p. Noworyta. *Nature, 2001, 414*, 188.

[22] Dresselhaus, M. S.; Dresselhaus, G.; Avouris, P. In *Carbon Nanotubes: Synthesis, Structure, Properties and Applications*; Springer Series in Topics in Applied Physics; vol. 80; Springer: Berlin, 2001.

[23] Harris, PFJ. *Carbon Nanotubes and Related Structures: New Materials for the 21st Century*; Cambridge University Press: Cambridge, UK, 1999.

[24] Gelb, L. D.; Gubbins, K. E.; Radhakrishnan, R.; Sliwinska-Bartkowiak, M. *Rep. Prog. Phys.* 1999, *62*, 1573.

[25] Hansen, J. P.; McDonald, I. R. *Theory of Simple Liquids, 2nd edition*; Academic: London, 1986.

[26] Cahn, J. W. *J. Chem. Phys.* 1977, *66*, 3667.

[27] Nakanishi, H.; Fisher, M. E. *Phys. Rev. Lett.* 1982, *49*, 1565.

[28] Frenkel, D.; Smit, B. *Understanding Molecular Simulation from Algorithms to Applications*. Academic press: London, 2002.

[29] C. Aldrich. Learning by Doing : *A Comprehensive Guide to Simulations, Computer Games, and Pedagogy in e-Learning and Other Educational Experiences.* Pfeifer — John Wiley & Sons: San Francisco, 2003.

[30] Charles, K.; Kroemer, H. *Thermal Physics, Second Edition.* W.H. Freeman and Company: San Francisco, 1980.

[31] Norman, G.E.; Filinov, V.S. *Investigation of phase transitions by a Monte Carlo method. High Temp*. (USSR), *1969*, *7*, 216.

Chapter 2

Methods and Simulation Code Validation

2.1 Introduction

The first molecular dynamics simulation of water was reported by Rahman and Stillinger in 1971[1]. They used a four-point-charge model and Lennard-Jones parameters to carry out a 2-ps simulation using a hard sphere molecular dynamics algorithm developed by Alder and Wainwright[2]. Around the same time, Barker and Watts reported the first Monte Carlo simulation study of the structural properties of liquid water[3]. Following these seminal publications, extensive simulations of water and aqueous solutions were performed on a wide range of systems[4-22]. These studies included many in which new water models were developed and evaluated under various conditions[6,7,8,9]. Other simulations described the structural, thermodynamic, kinetic properties for both water[10-13] and solutes[14-16] in aqueous solution, representing experimental results well[17,18]. Advances in material science over the past 10 years has been accompanied by water research focusing on aqueous solutions in porous materials[19-22] in which the water usually shows different structural properties compared with bulk water. Numerous Grand-Canonical Monte Carlo (GCMC) simulations of water have also been reported recently[23-27]. GCMC simulations yield directly the number of particles in the fluid phase of a system as a function of the water chemical potential which relates directly to the water vapor pressure[28,29]. GCMC simulation methods may

be applied to both pure water and to aqueous solutions as long as the fluid density is not too high. In particular, GCMC methods are well suited to the study of porous media including calculations of important quantities such as adsorption isotherms[21].

This chapter is organized as follows. At the beginning, the theory and procedures of GCMC simulations are described in detail including how the traditional Metropolis method is modified to determine acceptance ratio equations for molecule insertion and deletion steps. Results of GCMC simulations of pure water are then described. These simulations are used to demonstrate the efficiency of the code in generating fluctuations in the particle number and to optimize simulation parameters related to translational and rotational movement processes. Comparisons to previously published results are also used to validate the simulation code.

2.2 Theory and Method

2.2.1 Metropolis Monte Carlo theory

The Metropolis Monte Carlo (MC) method begins with the following statistical mechanical expression for the average value of some observable, A[28,29]:

$$\langle A \rangle = \frac{\int dr^N \exp[-\beta U(r^N)] A(r^N)}{\int dr^N \exp[-\beta U(r^N)]} \qquad (2.1)$$

where $A(r^N)$ is the observable in an N particle system, N is the number of particles, $\beta = 1/k_B T$ in which k_B is Planck's constant, and T is the temperature. The denominator is the so called "partition function" Q. The partition function is in

general impossible to evaluate directly. The Metropolis method in Monte Carlo simulations provides a scheme for exploring possible system configurations that generate the correct probability density $P(r^N)$ for finding the system in a configuration around r^N:

$$P(r^N) \equiv \frac{\exp[-\beta U(r^N)]}{\int dr^N \exp[-\beta U(r^N)]} \tag{2.2}$$

This allows for evaluation of averages in Eq. (2.1) through a summation over configurations sampled in a simulation:

$$\langle A \rangle = \frac{1}{L} \sum_{i=1}^{L} A(r_i^N) \tag{2.3}$$

where L is the total number of configurations sampled.

Each step in a MC simulation involves a "trial move" of a randomly selected particle. The key to the Metropolis method's generation of a proper probability density is related to the process by which trial moves are accepted or rejected. This process involves assuring that movement steps reproduce the condition of detailed balance upon movement between old (o) and new (n) configurations:

$$N(o)\alpha(o \to n) \times acc(o \to n) = N(n)\alpha(n \to o) \times acc(n \to o) \tag{2.4}$$

Where $N(o)$ and $N(n)$ are the probability densities for configurations *o* and *n*, respectively, while the *acc* variables reflect the acceptance probability for the trial move in the given direction. The "underlying matrix" α reflects the relative likelihood of attempting the particular trial move. It is usually a symmetric function:

$$\alpha(o \to n) = \alpha(n \to o) \tag{2.5}$$

Proper detailed balance can then be assured by adjusting the acceptance probabilities so that their ratio is follows:

$$\frac{acc(o \to n)}{acc(n \to o)} = \frac{N(n)}{N(o)} = \exp\{-\beta[U(n) - U(o)]\} \tag{2.6}$$

Any trial move for which the energy change $U(n) - U(o)$ is negative is accepted. Trial moves for which the energy change is positive are accepted with probability equal to $\exp\{-\beta[U(n) - U(o)]\}$. This process may be represented generally by the following relationship:

$$acc(o \to n) = \min\left(1, \exp\left\{-\beta\left[U\left(r_n^N\right) - U\left(r_o^N\right)\right]\right\}\right) \tag{2.7}$$

2.2.2 Grand Canonical Monte Carlo (GCMC) theory

In GCMC simulations, the system can be thought of as a cell in contact with a reservoir that imposes constant chemical potential and temperature constraints by exchanging particles and energy. The partition function of *N* interacting particles in volume *V* and *M*-*N* ideal gas molecules in volumes V_0-*V* is:

$$Q(N, MV, V_0, T) = \frac{V^N (V_0 - V)^{M-N}}{\Lambda^{3M} N!(M-N)!} \int ds^{M-N} \int ds^N \exp[-\beta U(s^N)] \tag{2.8}$$

Where V_0 is the total volume which includes the system cell and reservoir, *V* is the volume of system cell, *M* is the total particle number in both system cell and reservoir, *N* is the particle number in the system cell, *T* is the temperature, and the thermal de Broglie wavelength Λ is:

$$\Lambda = \sqrt{h^2 / (2\pi m k_B T)} \tag{2.9}$$

Where h is Planck constant and k_B is the Boltzmann constant. The corresponding probability density can be expressed as:

$$P(r^N, N) \propto \frac{\exp(\beta\mu N) V^N}{\Lambda^{3N} N!} \exp[-\beta U(r^N)] \tag{2.10}$$

For movement steps, the acceptance probability in GCMC simulations is the same as described above for MC simulations. For insertion and deletion trials, however, the acceptance ratio takes on the following form:

$$\frac{acc(N \to N+1)}{acc(N+1 \to N)} = \frac{\exp(\beta\mu) V}{\Lambda^3 (N+1)} \exp\left\{-\beta[U(r^{N+1}) - U(r^N)]\right\} \tag{2.11}$$

Where the system chemical potential μ depends on the particle density ρ:

$$\mu = k_B T \ln \Lambda^3 \rho \tag{2.12}$$

For bulk water, the chemical potentical reflects the vapor pressure directly. The energy difference in the exponential term of this equation reflects the energy of the system both with and without the inserted or deleted particle. A trial insertion or deletion is accepted if the term on the right hand side of Equation (2.6) is greater than one. Otherwise, it is accepted with a probability equal to the right hand side of that equation. For insertions, this may be expressed as:

$$acc(N \to N+1) = \min\left[1, \frac{V}{\Lambda^3 (N+1)} \exp\left\{\beta[\mu - U(N+1) + U(N)]\right\}\right] \tag{2.13}$$

It should be clear from this equation that increasing the chemical potential μ increases the likelihood of acceptance of insertion trials, and hence increases the

average number of particles in the system. Similarly, deletion trials are accepted with a probability given as follows:

$$acc(N \to N-1) = \min\left[1, \frac{\Lambda^3 N}{V}\exp\left\{-\beta\left[\mu + U(N-1) - U(N)\right]\right\}\right] \quad (2.14)$$

The GCMC process described to this point is valid for the case where the underlying matrix is symmetric:

$$\alpha(N \to N+1) = \alpha(N+1 \to N) \quad (2.15)$$

Where, as for MC simulations described above, the α functions represent the probability of attempting that particular trial. A significantly more efficient GCMC algorithm may be implemented, however, by allowing the underlying matrix to be asymmetric[30]. The computational expense for an insertion trial is significantly less than for a deletion trial since most insertions can be rejected quickly based on a rapid check for particle overlap. It is therefore beneficial to attempt far more insertions than deletions during the course of a simulation. This has no impact on the accuracy of the simulations as long as the resulting underlying matrix asymmetry is included in the acceptance probability expressions. Representing the ratio of insertion and deletion trials by the variable r_{id} yields the following acceptance probability equations[14]:

$$P_{acc}^{N \to N+1} = \min\left[1, \frac{V}{r_{id}\Lambda^3(N+1)}\exp[\beta(\mu - \Delta U)]\right] \quad (2.16)$$

$$P_{acc}^{N+1\to N} = \min\left[1, \frac{r_{id}\Lambda^3(N)}{V}\exp[-\beta(\mu+\Delta U)]\right] \tag{2.17}$$

In a previous study, it was found that a value of $r_{id} = 30$ was optimal[14].

2.2.3 Water model

The water model used in all simulations described in this thesis is the extended simple point charge (SPC/E) model of Berendsen and co-workers[31]. This model is a modified version of the widely used SPC model[32] that includes a self-polarization energy correction that is typically missing from empirical water models. The SPC/E model is known to reproduce well water's bulk density at atmospheric pressure and its diffusion constant[15]. It can also satisfactorily reproduce the structure of bulk water from room temperature to supercritical conditions[33,34]. A diagram representing the SPC/E water model is shown in Fig 2.1. Each water molecule consists of three interaction sites located at the atomic nuclei. The molecule has a rigid geometry with a bond length of 1 Å and a tetrahedral bond angle of 109.47°. The interaction potential for SPC/E water consists of a Lennard-Jones (LJ) potential[29] centered on the oxygen atom:

$$U(r) = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] \tag{2.18}$$

Where *r* is the distance between atoms, ε = 0.65 kJ/mol is the depth of the potential well, and σ = 3.166 Å is the (finite) distance at which the interparticle potential is zero. There are also partial charges located on each of the atomic sites,

a charge on oxygen of -0.8476 e and a charge on hydrogen of +0.4238 e[31].

2.2.4 Calculation of the system energy

The energy of a collection of SPC/E water molecules may be calculated, in principle, from a summation of all LJ and Coulomb interactions in the system:

$$U\left(r^N\right)=\sum_i\sum_j 4\varepsilon\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12}-\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right]+\frac{q_i q_j}{r_{ij}} \qquad (2.19)$$

where the summations are taken over all atom pairs except atoms within the same water molecule and the LJ potential term is zero except for oxygen-oxygen interactions. Periodic boundary conditions and the long range nature of Coulomb interactions, however, lead to significant complications in the proper calculation of the Coulomb terms in this expression.

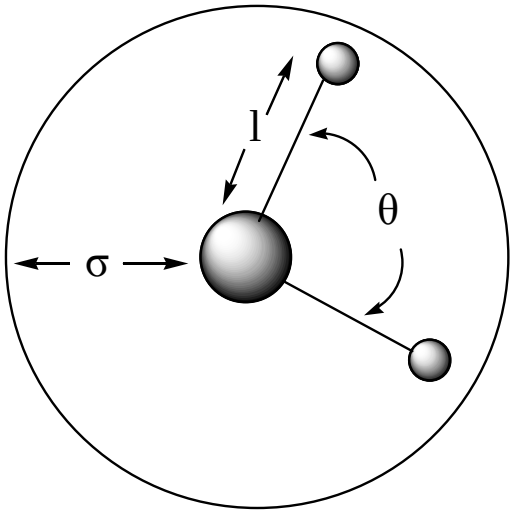


Fig. 2.1 Diagram of the SPC/E water model, l equals to 1.0 *Å*, q_1 is 0.4238e and q_2 is -0.8476e, σ equals to 3.166 *Å*, angle θ is 109.47°.

It is widely accepted that the best way to deal with these issues is through the Ewald summation method[35-37] that specifically accounts for interactions between the primary system cell and its infinite periodic images. In this method, each point charge in the system is first neutralized by a Gaussian charge distribution of opposite sign. This effectively shortens the range of the real-space Coulomb interactions to eliminate the need for interaction potential truncation. The effect of the additional Gaussian charge distributions is then corrected for by using another Gaussian charge distribution with opposite sign than the first distribution. Interactions among these correcting charge distributions are calculated in "reciprocal space" using a Fourier transform technique. The resulting Ewald summation formula may be expressed as:

$$\begin{aligned} U_{Coul} = & \frac{1}{2V}\sum_{k\neq 0}\frac{4\pi}{k^2}\left|\rho(k)\right|^2 \exp\left(-k^2/4\alpha\right) \\ & -(\alpha/\pi)^{\frac{1}{2}}\sum_{i=1}^{N} q_i^2 \\ & +\frac{1}{2}\sum_{i\neq j}^{N}\frac{q_i q_j erfc\left(\sqrt{\alpha} r_{ij}\right)}{r_{ij}} \\ & -\sum_{i=1}^{N}\sum_{k\neq l}^{M}\left(\frac{q_k q_l}{r_{kl}} - \frac{q_k q_l erfc\left(\sqrt{\alpha} r_{kl}\right)}{r_{kl}}\right) \end{aligned} \qquad (2.20)$$

where V is the volume of the box, $k = (2\pi/L)l$ with $l = (l_x, l_y, l_z)$ are the lattice vectors in Fourier space, L is the side length of the cube, $\rho(k)$ is the charge density, α is constant related to a Gaussian distribution parameter, N is the number of molecules in the system, erfc is the complementary error function, M is the number of atoms in

each molecule, q is atomic charge, and r is the distance between two particles[29]. The four terms in the Ewald summation expression, each represented as a separate line in Equation (2.19), are denoted (from top to bottom) as the reciprocal space term, the self term, the real space term, and the intramolecular term. The computationally expensive portion of the Ewald calculation is the reciprocal space term. It increases with the number of lattice vectors (k). The Gaussian parameter α is chosen carefully to adequately shorten the range of the real-space term without requiring an excessive number of lattice vectors for convergence of the reciprocal space term.

2.2.5 System setup

The system for simulation of bulk water is chosen to be a cubic, three-dimensional box with side length of 31.748 Å. Periodic boundary conditions in which a particle exiting from one side of the system will reappear from the opposite side are used to mimic a system of infinite extent. With respect to energy calculations, each molecule is allowed to interact in real-space only with another molecule's closest image. This is the standard "minimum image convention." Starting configurations for the water molecules were taken either from previous simulations or by starting with an empty system. The GCMC algorithm created a system of the same average density and structure in either case. All simulations were performed at a temperature of 298.15 K and a water chemical potential of -11.61 kcal/mol. This value of the chemical potential is known to give a bulk

water density close to the proper experimental value at 298.15K[14].

2.2.6 Generating a trajectory

Once model parameters and a starting configuration were selected, generation of a GCMC trajectory proceeded as follows. A random number is first used to select the type of trial to be considered. The probability of selecting a movement trial or a deletion trial is taken to be equal. As discussed above, insertion trials are selected with greater frequency than deletion trials since they are more computationally efficient. A value of $r_{id} = 30$ then gives a probability for selection of deletion or movement trials of 1/32 and the probability of selection of an insertion trial of 30/32.

If an insertion attempt is selected, a trial water molecule is created and placed into the system at a random location and with random orientation. An algorithm first checks for significant overlap with other water molecules since such high-energy configurations will never be accepted based on the criterion of Eq. (2.11). If no significant overlap is observed, the energy of the trial particle is calculated. The insertion is then accepted or rejected based on Eq. (2.13).

If a deletion attempt is selected, an existing water molecule is chosen at random and the energy change upon deletion is calculated. The deletion is accepted or rejected based on Eq. (2.14).

If a movement attempt is selected, an existing water molecule is chosen at random and subjected to a translation and rotation determined as follows. The

magnitude of the translation in each Cartesian direction is set to be

$$T_\alpha = T_{\max}(N - 0.5) \tag{2.21}$$

where α represents the *x*, *y*, or *z* coordinate, *N* is a random number on the interval between 0 and 1, and $T_{\max}$ is an adjustable parameter representing the maximum translation. Similarly, the magnitude of the rotation along the various possible rotational coordinates is set to be

$$R_\alpha = 2\pi R_{\max}(N - 0.5) \tag{2.22}$$

where α represents a quaternion rotational axis, *N* is again a random number on the interval between 0 and 1, and $R_{\max}$ is an adjustable parameter representing the maximum magnitude of rotation in radian units.

Once a trajectory is initiated, the approach to equilibrium is assessed by convergence of the total energy and the particle number. At equilibrium, the particle number should fluctuate around some average value as shown in Fig. 2.2. Each "cycle" in the figure refers to a total of (r_{id} + 2) attempted move, insertion, or deletion events. In other words, one cycle consists of, on average, one attempted move, one attempted deletion, and r_{id} attempted insertion events.

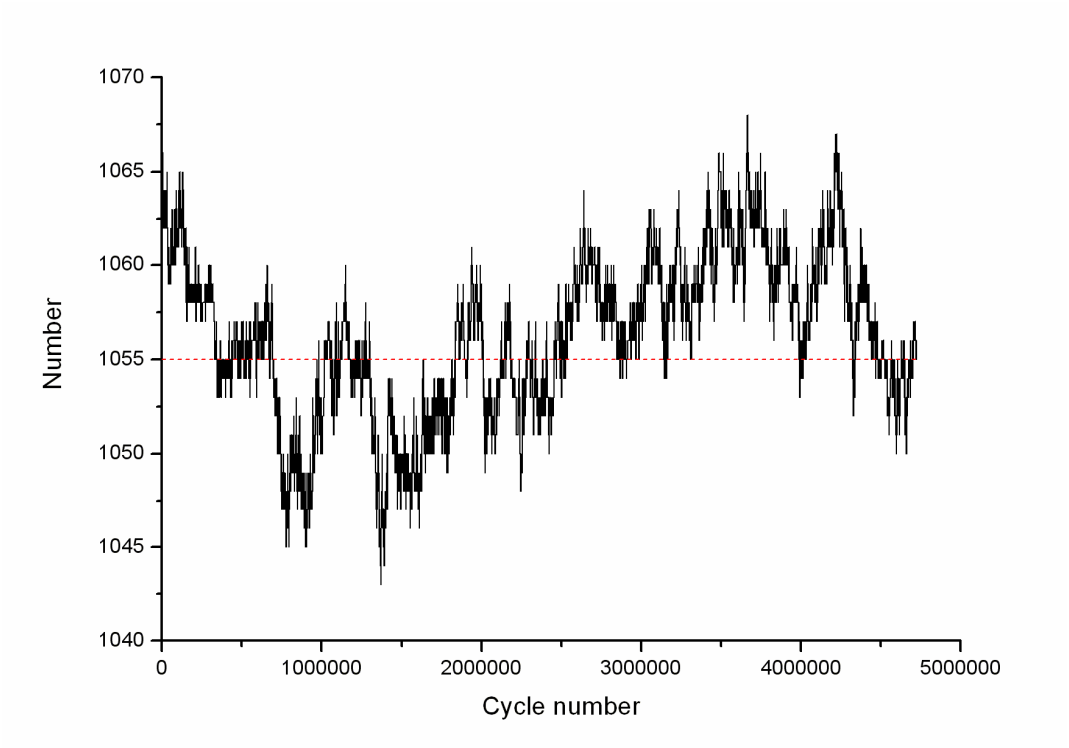


Fig. 2.2 Fluctuations in the water particle number in the system.

2.2.7 Optimization of movement parameters

The adjustable parameters T_{max} and R_{max} described above should be optimized to achieve the most efficient sampling of system configurations. To achieve this, we first introduce a quantity called the Monte Carlo displacement (MCD) that is analogous to the mean square displacement used in molecular dynamics simulations to determine diffusion coefficients. The MCD is defined as

$$\mathrm{MCD}(i) = \left\langle \left| \vec{r}(n) - \vec{r}(n-i) \right|^2 \right\rangle \qquad (2.23)$$

where *n* and *i* refer to the number of MC cycles performed and *r(i)* refers to the oxygen atom position of a tagged particle after *i* MC cycles of the simulation. An example of an MCD plot is shown in Fig. (2.3.1). The slope of the graph is initially large but then decreases to a relatively constant value after 500,000 MC cycles. It is the limiting slope or, alternatively, the value of MCD*(i)* after a large number of MC cycles that should be maximized (by variation of the T_{max} and R_{max} parameters) in order to obtain the most efficient particle motion possible.

The efficiency of rotational motion may be assessed by calculation of a rotational correlation function (RCF)

$$\mathrm{RCF}(i) = \left\langle \overrightarrow{\mathbf{R}_{\mathrm{OH}}}(n) \bullet \overrightarrow{\mathbf{R}_{\mathrm{OH}}}(n-i) \right\rangle \qquad (2.24)$$

where $\overrightarrow{\mathbf{R}_{\mathrm{OH}}}$ refers to an O-H vector in a tagged water molecule. The dot product in the equation is equal to 1 at $i = 0$ as the two unit vectors point in the same direction initially, but decays eventually to zero as the orientation of the molecule becomes randomized. A sample plot of the RCF function is shown in Fig. (2.3.2). Full

orientational randomization after which RCF*(i) = 0* clearly takes longer than the 5 x 10^5 MC cycles plotted here. Nevertheless, simulations of this length should be adequate to achieve optimization of the R_{max} parameter. The full procedure for optimization of the T_{max} and R_{max} parameters will be described in the Results section below.

2.2.8 Pair distribution function g(r)

The pair distribution function[38,39] g(r) reflects the probability of finding a pair of atoms a distance *r* apart and is therefore a measure of the system structure. It is defined as:

$$g(r) = \frac{V}{N^2}\left\langle \sum_i \sum_{i \neq j} \delta\left(r - r_{ij}\right) \right\rangle \tag{2.25}$$

where *V* is the volume, *N* is the particle number, and $\delta(r - r_{ij})$ is a Kroniker δ function. The pair distribution function may be calculated for any pair of atom types, i.e., $g_{OO}(r)$ represents the oxygen-oxygen pair correlation function while $g_{OH}(r)$ represents the oxygen-hydrogen pair correlation function, etc.

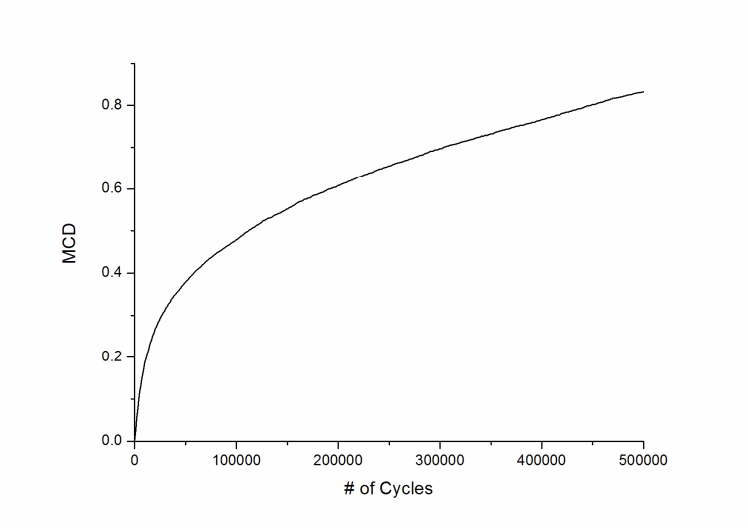


Fig. 2.3.1 MCD vs cycle number by the optimized parameters

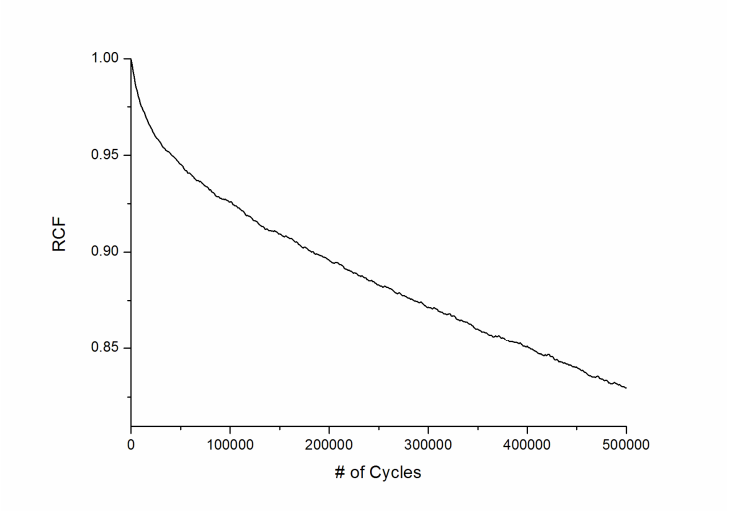


Fig. 2.3.2 RCF vs cycle number by the optimized parameters

2.3 Results and Discussion

2.3.1 Movement parameter optimization

Movement parameter optimization simulations were run using the MC rather than the GCMC simulation code since particle insertion and deletion events would interfere with calculation of the MCD and RCF functions. MC simulations for each set of T_{max} and R_{max} parameters considered were run for a total of 10^6 cycles with water position and orientation information stored every 1000 cycles. The MCD(i) and RCF(i) functions were each calculated out to a value of i = 5 x 10^5 cycles. Optimization was performed sequentially for the T_{max} and R_{max} parameters. For optimization of T_{max}, the value of R_{max} was set to be 0.07 based on values used previously in our research group. The value obtained for MCD(*i*) at *i* = 5 x 10^5 is plotted as a function of T_{max} in Fig. 2.4.1. The MCD plot shows a peak at a value of $T_{max} = 0.5$. For smaller values, the translation step size is too small for efficient displacement even though a large fraction of the trial moves are accepted. For larger values of T_{max}, a significant drop in the acceptance ratio has a detrimental impact on the displacement.

R_{max} values ranging from 0.03 to 0.17 were then investigated with the value of T_{max} set to 0.5. Results are displayed in Fig. 2.4.2. The lower panel of the figure shows that the value of RCF(*i* = 1 x 10^5) is reduced steadily as R_{max} increases up to around 0.13. Since the smaller values for the RCF are better, a value in the range of 0.11 to 0.13 seems optimal. However, increasing R_{max} eventually has a

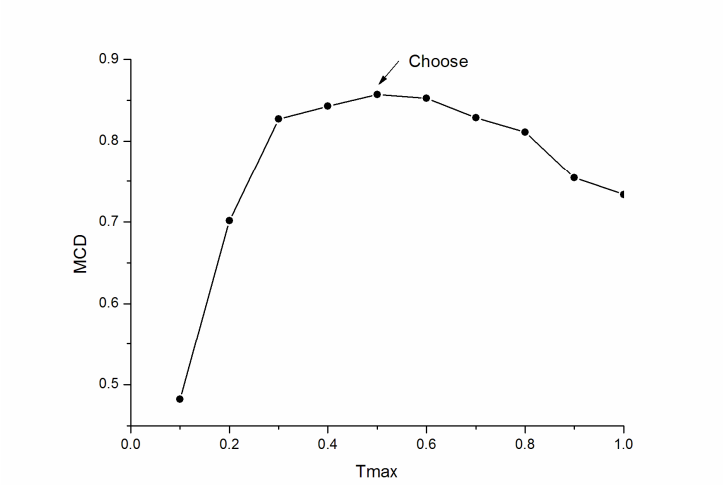


Fig. 2.4.1 Plot of MCD(i = 5 x 10^5) vs T_{max} with R_{max} = 0.06. A value of T_{max} = 0.5 was chosen as it gave the largest value for the MCD.

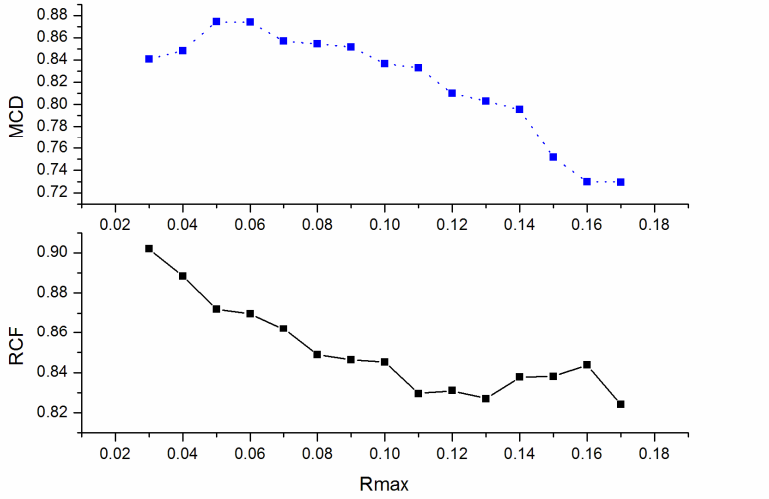


Fig. 2.4.2 Plot of RCF(i = 5 x 10^5) versus R_{max} (lower panel) with T_{max} = 0.5. The upper panel shows a plot of MCD(i = 5 x 10^5) versus R_{max} to illustrate how the two parameters are not independent.

detrimental effect on the value for the MCD function. This results from a reduction in the acceptance ratio for trial moves as the value of R_{max} gets larger. Values of R_{max} = 0.11 and T_{max} = 0.5 were therefore selected for use in the simulations. However, the fact that R_{max} affects both the MCD and RCF functions suggests that this optimization procedure cannot identify a single, single optimal set of values for T_{max} and R_{max}.

2.3.2 Program validation

The code used for all simulations in this thesis was written in the C++ programming language as a "translation" of an existing Fortran simulation code. The motivation for this change in programming language is discussed in Chapter 5. A significant effort in this translation project involved validation of the new C++ code through comparison with results from the existing, highly tested Fortran code. For bulk water simulations, this validation process involved comparison of energetic and structural properties as well as average densities determined from lengthy GCMC simulations.

The system energy was compared for the two codes in terms of (i) individual system configurations, (ii) trial insertion and deletion events, and (iii) lengthy averages over independent trajectories. All results compared favorably apart from small differences due to a numerically approximate procedure used for calculating inter-particle energies in the Fortran code. Average water density at a chemical potential of -11.61 kcal/mol and a temperature of 298.15 K was etermined

from the two codes for a cubic system with a side length of 23 Å. Results for the C++ code (1.010 g/cm^3) were in reasonable agreement with the density of 1.000 g/cm^3 generated from the Fortran code[14], especially given the extensive simulation time required for convergence of the average density. Finally, structural results from the two codes also agree well, as seen in the graphs of $g_{OO}(r)$ displayed in Fig. 2.5.

2.3.3 Pair correlation functions for water

A complete set of pair correlation functions for water including $g_{OO}(r)$, $g_{OH}(r)$, and $g_{HH}(r)$ are displayed in Fig. 2.6. Based on the $g_{OO}(r)$ function, nearest neighbor water molecules in the system are separated by around 2.75 Å. This is somewhat smaller than the LJ σ parameter for SPC/E water, indicating that hydrogen bonding causes a decrease in the O-O distance. The second and third peaks in $g_{OO}(r)$ are located at 4.37 Å and 6.84 Å, respectively. Beyond this distance the g(r) function is nearly flat with a value of 1, as expected when atom-atom radial correlations are no longer significant. The first peak positions for the $g_{OO}(r)$ and $g_{OH}(r)$ functions are separated by nearly 1 Å, the O-H bond length, indicating that hydrogen bonds in the system are close to linear. Each water molecule has nearly four hydrogen bonds on average. The average energy of the hydrogen bonds (~20kcal/mol) is significant and stablizes the bulk water structure. Finally, the $g_{HH}(r)$ function indicates that the first peak is at 2.36 Å which is between the distance for the first peak of the O-O and the first peak of O-H

interactions. This is expected given the roughly tetrahedral structure of extensively hydrogen bonded water molecules.

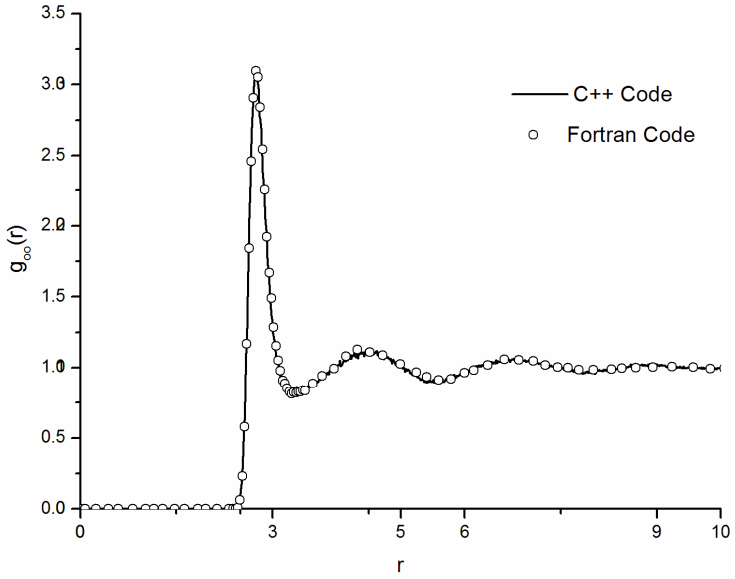


Fig. 2.5 Comparison of $g_{oo}(r)$ correlation functions generated by the C++ code (solid line) and Fortran code (circles).

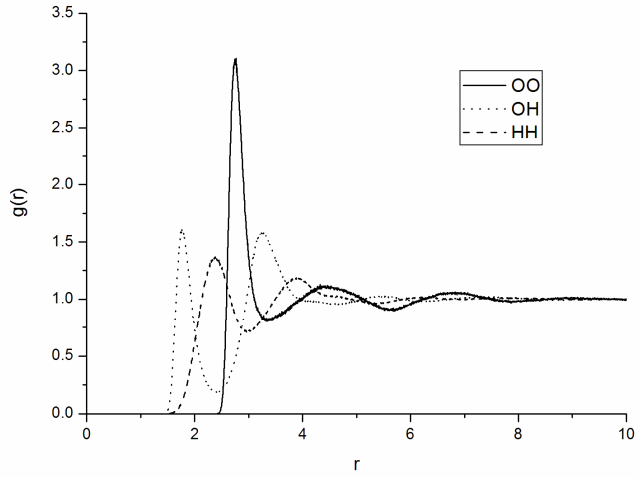


Fig. 2.6 Pair correlation functions for water.

2.4 References

[1] Rahman, A.; Stillinger, F.H. *J. Chem. Phys. 1971, 55,* 3336.

[2] Alder, B.J.; Wainwright, T.E. *J.Chem. Phys. 1959, 31*, 459.

[3] Barker, J.A.; Watts, R.O. *Chem. Phys. Lett. 1969, 3,* 144.

[4] Clementi, E; Corongiu, G. *Advances in biophysics. 1985, 20,* 75.

[5] Ma, B.; Nussinov, R. *Proteins. 1999, 37(1),* 73.

[6] Wallqvist, A.; Mountain, R.D. In Reviews in Computational Chemisry; Wiley-VCH: New York, 1999.

[7] Svishchev, I.M.; Kusalik, P.G.; Wang, J.; Boyd, R.J. *J.Chem. Phys. 1996, 105,* 4742.

[8] Chen, B.; Xing, J.H.; Siepmann, J.I. *J. Phys. Chem. B 2000, 104,* 2391.

[9] Wallqvist, A.; Berne, B.J. *J. Phys. Chem. 1993, 97,* 13841.

[10] Yonetani, Y. *Journal of Chemical Phycs. 2006, 124,* 104501.

[11] Hu, H.; Lu, Z.; Elstner, M.; Hermans, J; Yang, W. *Journal of Physical Chemistry A. 2007, 111,* 5685.

[12] Kumar R.; Skinner, J.L. *Journal of Physical chemistry B. 2008, 112,* 8311.

[13] Allesch, M. Schwegler, E. Gygi, F. Galli, G. *Journal of Chemical Physics. 2004, 120,* 5192.

[14] Whitley, H.D.; Smith, D. E. *Journal of Chemical Physics 2004, 120,* 5387.

[15] Smith, D.E.; Dang, L.X. *Journal of Chemical Physics. 1994, 100,* 3757.

[16] Hermida-Ramon, J.; Karlstroem, G. *Theochem. 2004, 712,* 167.

[17] Narten, A.H.; Levy, H.A. In Water A Comprehensive Treatise; Franks, F., Ed.; Plenum Press: New York, 1972.

[18] Narten, A.H.; Thiessen, W.E. *Science 1982, 217*, 1033.

[19] Pertsin, A.J. Grunze, M. *Langmuir*. *2000, 16*, 8829.

[20] Trouw, F.; Iton, L.E.; Davis, M.E. *Studies in Surface Science and Catalysis*. *1994, 84*, 851.

[21]Do, D.D.; Do, H.D. *Adsorption Science & Technology. 2003, 21*, 389.

[22] Watanabe, A.; Iiyama, T.; Kaneko, K. *Chem. Phys. Lett. 1999, 305*, 71.

[23] Heffelfinger, G.S.; van Swol, F. *J. Chem. Phys. 1994, 100*, 7548.

[24] Cao, D.P.; Wu, J.Z. Langmuir 2004, 20, 3759.

[25] Kaneko, K.; Ohba, T.; Ohkubo, T.; Utsumi, S.; Kanoh, H.; Yudasaka, M.; Iijima, Sumio *Adsorption, 2005, 11*, 21.

[26] Thomson, K.T.; Gubbins, K.E. *Langmuir 2000, 16*, 5761.

[27] Wongkoblap, A.; Do, D.D. *J. Phys. Chem. B 2007, 111*, 50.

[28] Chandler, D. *Introduction to Modern Statistical Mechanics*; Oxford University press: 1987.

[29] Frenkel, D.; Smit, B. *Understanding Molecular Simulation from Algorithms to Applications*. *2002*, Academic press: London.

[30] Shelley, J.C.; Patey, G.N. *J. Chem. Phys. 1995, 102*, 7656.

[31] Berendsen, H.J.C.; Grigera, J.R.; Straatsma, T.P. *J. Phys. Chem. 1987, 91*, 6269.

[32] Berendsen, H.J.C.; Postma, J.P.M.; van Gunsteren, W.F.; Hermans, J. Intermolecular Force, Reidel, Dordrecht, 1981.

[33] Chialvo, A.A.; Yezdimez, E.; Driesner, T.; Cummings, P.T. Simonson, J.M. *Chem. Phys. 2000, 258*, 109.

[34] Hayward, T.M.; Svishchev, I.M. *Fluid Phase Equilibr. 2001, 182*, 65.

[35] Allen, M.P.; Tildesley, D.J. Computer simulation of liquids; Clarendon Press and Oxford University Press: New York, 989.

[36] York, D.M.; Darden, T.A.; Pedersen, L.G. *J. Chem. Phys. 1993, 99,* 8345.

[37] Darden, T.; Perera, L.; Li, L.P.; Pedersen, L. *Struct. Fold. Des. 1999, 7*, R55.

[38] Ciccotti, G.; Frenkel, D.; McDonald, I.R. *Simulation of Liquids and Solids Molecular Dynamics and Monte Carlo Methods in Statistical Mechanis. 1987*, North-Holland, Amsterdam.

[39]Hansen, J.P.; McDonald, I.R. *Theory of Simple Liquids 2nd edition. 1986 ,* Academic press: London.

Chapter 3

Hydrated carbon nanotubes

3.1 Background

The discovery of carbon nanotubes (CNTs) is generally attributed to Sumio Iijima in 1991[1] even though some other groups reported hollow structures of carbon filaments before that date.[2-4] Single-walled CNTs were subsequently discovered by Iijima[5] and Bethune[6] in 1993. Following their discovery, the study of CNTs quickly became one of the most active and exciting areas of research in materials physics and chemistry. Over the past twenty years there have been over forty thousand papers published related to CNTs. These include investigations of their remarkable structural, mechanical and electromechanical properties[7-9] and their use in a wide range of applications such as gas storage,[10-13] nanoelectronics,[14-15] molecular detection,[16-17] membrane separation,[18,19] nanopipets,[20] nanotweezers,[21] gases or liquids delivery,[22,23] and drug delivery.[24,25]

Hydrated CNTs are of significant theoretical interest in chemistry, biology and materials science[26,27]. They constitute a class of simple, variable-diameter, porous materials that may mimic the water-filled porous structures that are present in biological cells, membranes, proteins[28], and various catalytic materials[29]. Several computational models have been developed[30,31,32] to represent hydrated CNTs under extreme conditions or in cases where experimental observation is difficult to obtain. Most of the simulations performed to date concentrate on

single-walled CNTs as these are simpler than multi-walled CNTs and also occur commonly in CNT materials. The first computational investigation of hydrated CNTs was published in 2000[33]. Many others followed shortly leading to the accumulation of a significant amount of molecular-level information about hydrated CNT behavior[34].

Water density distributions both inside and outside of CNTs have been studied by different groups[12,32,35,36]. For narrow CNTs, water forms a linear chain inside the hydrophobic CNT channel, resulting in a net loss of hydrogen bonds compared with water in bulk water. This is compensated for somewhat by the van der Waals attraction to the nanotube walls[35] and also by a strengthening of the individual hydrogen bonds. It has been shown furthermore that entry of a single water molecule into the CNT channel is thermodynamically unfavorable, and that formation of a complete water chain is required in order for filling to be advantageous[37]. In larger diameter CNTs, water still forms highly ordered, layered structures that may involve n-gonal rings[38-40] in addition to linear chains[34]. Studies of the dynamics of water flow through CNTs have also revealed interesting cooperative behavior along water molecule chains as well an important role for defect structures[31,35].

Recently, simulations of functionalized CNTs have also been reported.[41,42] In these studies, ions were anchored on the inner wall or outside wall of a CNT[43,44]. They showed, for example, that it is possible to manipulate the structure of water

within CNTs by varying the concentration and location of these functional groups[37]. It is even possible to change the CNTs to exhibit hydrophilic rather than hydrophobic properties[45]. In general, functionalization opens of a very broad range of applications involving CNTs that would not otherwise be feasible.

Trichloromethane, more commonly known as chloroform, is a common by-product of industrial chemical reactions[46]. It was named and chemically characterized in 1834 by Jean-Baptiste Dumas[47]. Chloroform was commonly used as a general anesthetic during childbirth during the 19th century, but this use and others were eventually abandoned due to its toxicity. Chloroform has been banned as a consumer product in the United States since 1976[48]. The United States National Institute for Occupational Safety and Health has reported that breathing about 900 ppm chloroform can cause dizziness, fatigue, and headache. It will depress the central nervous system and cause damage to the liver and kidneys[46]. It can also cause damage to various plants including brittleness in roots and chromosomal damage. Because it can dissolve modestly in water and does not readily bind to soil minerals, chloroform can travel down through soil to groundwater where it can enter a water supply and persist for long times due to the absence of light and air. The breakdown products of chloroform in air include phosgene which is more toxic than chloroform itself[49]. People can encounter chloroform by drinking water or beverages made using water containing chloroform. These reasons lead to the recognition of chloroform as a significant

environmental hazard.

Current technologies associated with chloroform pollution prevention and remediation focuses either on combustion processes[49] or adsorption methods[50]. Because there are no natural sources for chloroform, reduction of industrial releases of chloroform is a primary pollution control strategy. Specifically, current pollution control measures target chloroform release from industrial effluents, municipal waste treatment plant discharges, hazardous waste sites, and sanitary landfills and spills[50]. According to the Federal Resource Conservation and Recovery Act (RCRA)[49], the ultimate disposal of chloroform, preferably mixed with other combustible fuels, can be accomplished by controlled incineration. The drawbacks of this method involve the need for acid scrubbing of combustion gases as well as the large energy consumption associated with incineration. Alternatively, V_2O_5 catalysts that aid in the oxidization of chloroform to CO_2 and HCl[51] may provide a viable alternative to incineration. Chemical redox methods[52] have also shown significant promise in this regard. For chloroform previously released in the environment, adsorption has often been the method of choice in remediation, particularly in drinking water purification. For example, many materials have been investigated as chloroform adsorbents in recent years including fiber[53], zeolites[54], activated carbon[55], fullerenes[50]. For the most part, however, these studies have been restricted to the laboratory level, with none of the adsorbents displaying exceptional affinity for chloroform adsorption.

Investigation of alternative adsorption materials is therefore warranted. CNTs have shown significant potential as a possible alternative material[56], although chloroform studies to date are limited to one investigation of gas-phase adsorption[57].

In this chapter, results of a series of simulations of hydrated single-walled CNTs are reported with a goal of characterizing chloroform adsorption to the CNT surface. The term "CNT" will from this point refer specifically to single-walled CNTs since these are the only CNTs considered in our simulations. Results from simulations of pure water in CNTs are first presented. Calculated structural and binding energy distribution functions are compared with published results in order to validate the simulation program. Next, results of simulations of high-concentration chloroform solutions are presented with a focus on chloroform clustering near the CNT surface. The distribution of chloroform as a function of distance from the CNT axis was measured for two solutions with varying chloroform concentration. The binding free energy profiles were calculated for a single chloroform molecule and compared with binding free energies from the high concentration simulations. Finally, orientational distribution functions for chloroform were measured as a function of distance from the CNT axis.

3.2 Theory and Method

3.2.1 Building the CNT System

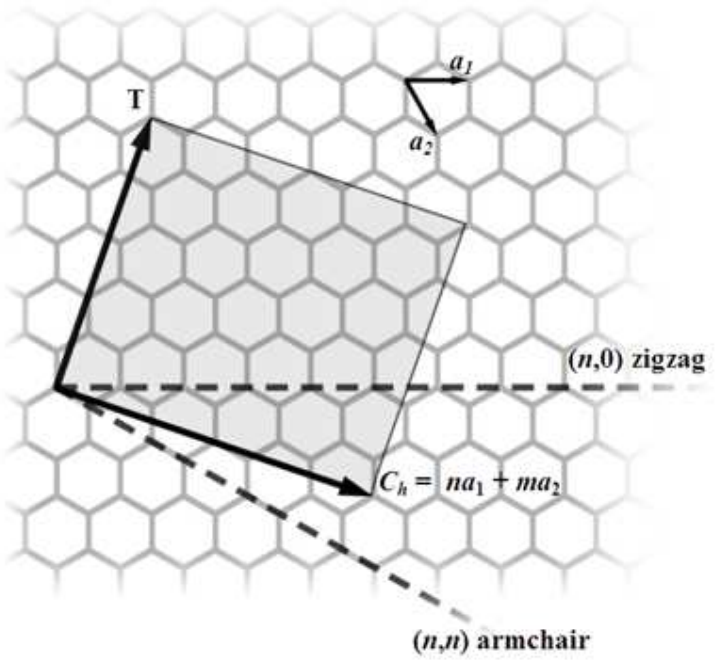
Simulation coordinates for the carbon atoms in a single-walled CNT are typically derived from carbon atom coordinates in a single graphite (i.e., "graphene") sheet through a process of rolling the sheet into a cylinder. The direction for rolling the graphene sheet may be represented by the chiral vector (*n*, *m*) (Fig 3.1a) where the integers *n* and *m* denote the number of unit vectors along two directions in the lattice of graphene. A nanotube formed by rolling along a vector with $m = 0$ is usually called a "zigzag CNT", while one formed by rolling along an $n = m$ vector is called an armchair CNT. Other types are called "chiral CNTs"[58]. A unit cell for the formation of a (6, 6) armchair CNT is displayed in Fig. 3.1b. The C-C bond length in a graphene sheet is 1.41 Å. Rolling of the (6, 6) graphene sheet therefore yields a CNT with a diameter of 7.90 Å, as shown in Fig. 3.2.1. A Fortran code for generating coordinates for any desired CNT is attached in Appendix B. All CNTs in the simulations reported below are of the single-walled (6,6) variety.

Once its coordinates are generated, the CNT is placed in a cubic system of side length 31.75 Å with its axis oriented in the z direction. The box length is a multiple of the CNT unit cell length. Periodic boundary conditions therefore generate a CNT that is effectively infinite in length. A system containing two CNTs oriented perpendicularly to each other is also created for some simulations.

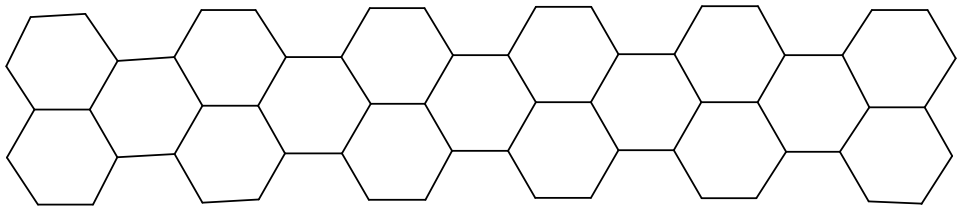
The minimum distance between C atoms in the two CNTs is taken to be 2.50 Å. An image of this system is shown in Fig. 3.2.2. Once the CNT coordinates are determined, hydrated and equilibrated CNT systems are generated using the GCMC simulation code described in Chapter 2 starting with an initially empty system.

3.2.2 Building the hydrated chloroform/CNT system

Initial configurations for the hydrated chloroform/CNT systems were generated as follows. Starting with an equilibrated, hydrated CNT system, the desired number of chloroform molecules (either 20 or 50) were inserted at random positions and orientations outside of the CNT. The (6, 6) single-walled CNT structures considered here are too narrow to contain chloroform. The GCMC method described in Chapter 2 was then used to equilibrate the system, with deletion, insertion and movement events all considered for water molecules, with the water chemical potential chosen to be the pure bulk water value of -11.61 kcal/mol. This provided for rapid removal of water molecules in overlapping configurations with inserted chloroform molecules. The chloroform content of the system was held constant, but MC movement trials were used to equilibrate the system with respect to chloroform positions. The CNT particle was held in a fixed position throughout the simulations, and the temperature was maintained at a constant value of 298.15K.



(a)



(b)

Fig. 3.1 The (*n*, *m*) nanotube naming scheme (a); Graphene layer for (6, 6) single walled carbon nanotube (b).

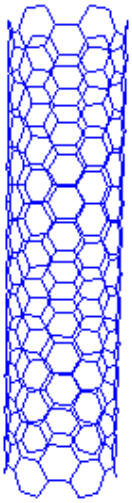


Fig. 3.2.1 (6, 6) Single-walled carbon nanotube.

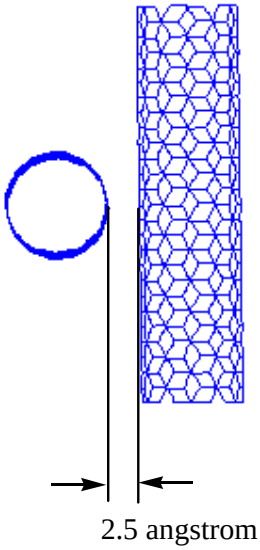


Fig. 3.2.2 Perpendicular double carbon nanotubes.

3.2.3 Interaction model

Chloroform molecules in the simulations are represented by the four-site model of Abrahamson and Wiles.[59] Three of the sites represent the Cl atoms while the fourth site represents a unified C-H group (Fig. 3.3). As is typical in hydrocarbon simulations, the C and H atoms are combined and represented as a single site since the hydrogen atom is small. Throughout this chapter the combined CH site will be represented as simply "C". The C-Cl bond length in the model is 1.758 Å and the Cl-C-Cl bond angle is 111.3°. Site-site interactions in this model are of the Lennard-Jones plus Coulomb variety as described for water in Chapter 2. Parameters for the model are given in Table 3.1 which also summarizes the parameters for both the CNT and the SPC/E water models. For the purpose of MC movement steps, the CH group is considered the center of the molecule about which rotational trial movements are performed.

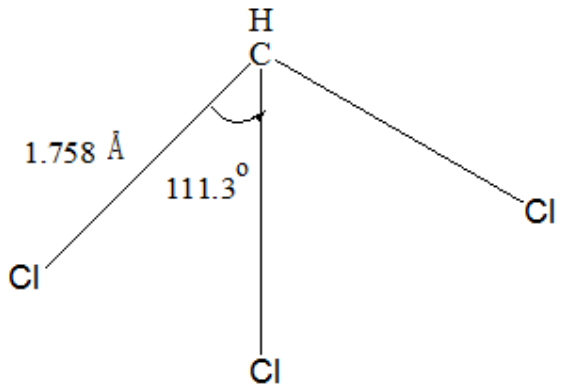


Fig. 3.3 Chloroform molecule model.

Table 3.1 Simulation parameters for Chloroform, Water and CNT

Molecule Type	Atomic Site	σ(Å)	ε(kcal/mol)	q(e)
Chloroform	CH	3.8	0.08	0.42
	Cl	3.47	0.4	-0.14
Water	O	3.17	0.65	-0.8476
	H	-	-	0.4238
CNT	C	3.4	0.05564	0

3.2.4 MC movement parameter optimization

Optimization of the MC translation and rotation step sizes for both water and chloroform in the CNT systems containing 50 chloroform molecules was performed using a procedure similar to that described in Section 2.2. Parameters for water were not re-optimized from their bulk-water values, but were checked following determination of the chloroform parameters and found to be affected only slightly by the presence of the chloroform solute and the CNT. The water parameters T_{max} and R_{max} were fixed at their bulk water values of 0.5 and 0.11 as determined in Chapter 2. A translational step size T_{max} 0.6 for chloroform was determined to be the best when we investigated T_{max} over a range 0.1 to 1.1 Å with a fixed rotational step size R_{max} of 0.07.

3.2.5 Binding energy

Binding energy distribution functions are useful in characterizing the different environments for water located either inside or outside of the CNT. The instantaneous binding energy u of a given water molecule is defined as the potential energy difference of the system in a given configuration with and without that molecule[34]. The probability distribution for the binding energy, $P_{bind}(u)$, is related to the excess chemical potential (μ^{ex}) through the relationship:

$$\exp\left(\beta\mu^{ex}\right) = \left\langle \exp\left(\beta u\right)\right\rangle = \int P_{bind}\left(u\right)\exp\left(\beta u\right) du \qquad (3.1)$$

Where $\beta = 1/k_BT$, k_B is Boltzmann's constant, and T is the temperature. The probability distribution function may be calculated readily from any MC simulation

as molecular energies are calculated regularly during the MC trajectory.

3.2.6 Structural Characterization

The structure of the hydrated chloroform/CNT system was characterized in terms of radial distribution functions (RDFs) for chloroform and water with respect to the CNT radial axis. These calculations are similar to standard RDF calculations as described in Chapter 2 (Eq. 2.24) except that the distance calculated is the distance from the CNT axis in the x-y plane. The water-CNT and chloroform-CNT RDFs are denoted $g_{O\text{-}CNT}(r)$ and $g_{C\text{-}CNT}(r)$, respectively.

Chloroform orientation with respect to the CNT is defined in terms of the vector B, displayed in Fig. 3.4, pointing from the chloroform C atom perpendicularly to the plane of the three Cl atoms. The angle θ is then defined as the angle between the vector B and the vector A pointing from the CNT axis toward the chloroform atom (see Fig. 3.4). The orientational distribution of chloroform molecules is then characterized in terms of a $\cos(\theta)$ distribution function. A value $\cos(\theta) = 1$ corresponds to the Cl atoms pointing away from the CNT, while a value of $\cos(\theta) = -1$ corresponds to the Cl atoms oriented toward the CNT.

3.2.7 Reversible work and force

The function $g_{C\text{-}CNT}(r)$ is directly related to the Helmholtz free energy of association, w(r), between a chloroform molecule and the CNT. The Helmholtz free energy is equivalent to the reversible work, w(r), for moving the particle to a position r from a starting point infinitely far from the CNT. The relationship

between g(r) and w(r) is given by[60]

$$g(r) = e^{-\beta\omega(r)} \tag{3.2}$$

where as noted previously is equal to $1/k_BT$ and k_B is Boltzmann's constant. The function w(r) is also often called the potential of mean force (PMF) which describes an average over all the conformations of the surrounding solvent molecules for two particles with distance *r*. This relationship is shown in equation (3.3):

$$\frac{\partial w(r)}{\partial r} = \langle F(r) \rangle = \left\langle \frac{\partial u(r)}{\partial r} \right\rangle \tag{3.3}$$

With the hydrated system, the PMF reflects the solvent-induced interaction between a chloroform molecule and the CNT as well as the intrinsic interaction between those two particles.

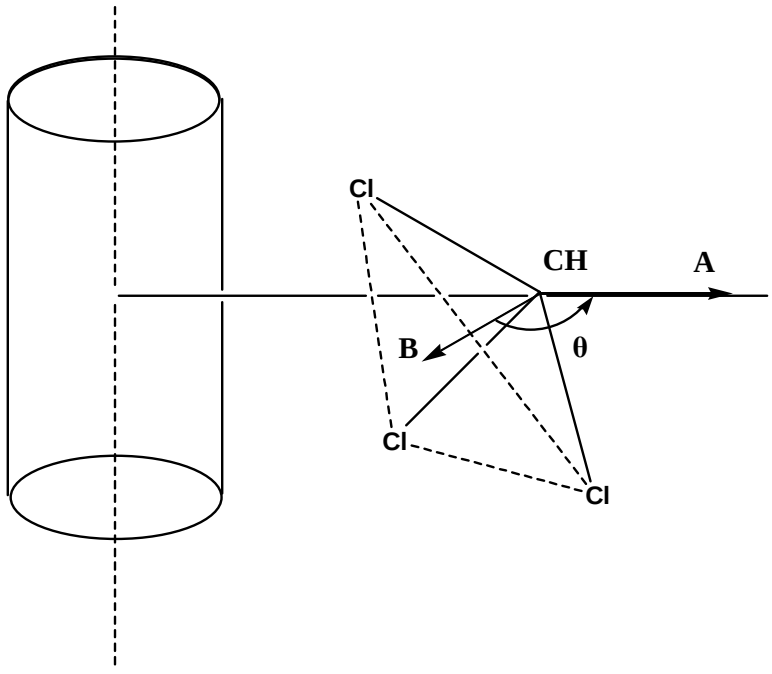


Fig. 3.4 The orientation angle θ.

In the system with 20 or 50 chloroform molecules, the correlation function g(r) may be determined by averaging over the observed structures. The PMF function may then be obtained by equation (3.2). However, for systems containing a single solute particle, an alternative approach is required in order to obtain a reasonable statistical average. In this case, w(r) may be determined conveniently by performing the average on the right side of equation (3.3)[54]. The chloroform solute molecule is held at a series of fixed distances from the CNT with the average force is calculated at each position. Rotational motion of the chloroform molecule is not constrained during this process. The calculated "mean force" is then integrated to yield the PMF, w(r). The partial derivative on the right hand side of equation (3.3) is evaluated using a finite difference method as $\Delta u / \Delta r$ with a value of Δr =0.01 Å. The mean force is calculated from 7.0 to 12 Å by step 0.1 Å.

3.2.8 Chloroform Solvation Free Energy

The Helmholtz free energy of solvation for chloroform may be determined using a thermodynamic integration (TI) method in which a chloroform molecule is inserted gradually into the system[61]. Chloroform is too large to be efficiently inserted in a single step using the GCMC method described in Chapter 2 or the closely related Widom method[62]. A computationally feasible gradual insertion is accomplished by defining a coupling parameter that varies from zero to one and, through that variation, "turns on" the desired interaction potential. This ransformation between interaction potentials may be expressed mathematically as:

$$\phi(\lambda) = (1-\lambda)\phi_A(r) + \lambda\phi_B(r) \tag{3.4}$$

where $\phi_A(r)$ and $\phi_B(r)$ are the interaction potentials for the initial (small molecule or ideal gas) and final (chloroform or chloroform precursor) states.

Once the coupling parameter λ is defined, the TI method consists of evaluating the following relationship[61]:

$$\Delta A_{ij} = \int_{\lambda_i}^{\lambda_j} d\lambda \left\langle \frac{\partial U(\lambda)}{\partial \lambda} \right\rangle_{\lambda} \tag{3.5}$$

where the values of λ again vary from zero to one and the variable *A* is the Helmholtz free energy. This method is analogous to the PMF calculation described in equation (3.3) except that the partial derivative "force" is in this case is along the λ coordinate rather than the spatial coordinate *r*.

In these simulations, a single chloroform molecule was grown from a non-interacting point particle in two stages as shown in Fig 3.5. The first stage involves growing a small, uncharged atom with a Lennard-Jones type interaction potential with CH LJ parameters from a non-interacting point particle. The second growth stage begins from the fully-grown uncharged atoms. Three chlorine atom sites are defined. The C-Cl bond length and the Cl interaction potential are both zero at λ = 0 and grow to their full values at λ = 1. The particle was constrained to be located a distance *r* = 7.0 Å from the CNT axis through both stages of the simulation. Its orientation was constrained to be parallel with the CNT during the second growth stage ($\cos\theta = 0$). A step size of λ = 0.1 was used

for both stages.

In order to validate the simulation code, the TI method was also used to calculate the excess chemical potential, μ_{excess}, for argon in bulk water, a quantity that has been determined elsewhere[61]. Details are not given here except to note that the resulting value for μ_{excess} of 2.3 kcal/mol is in reasonable agreement with the value of 2.07 kcal/mol from Ref.61.

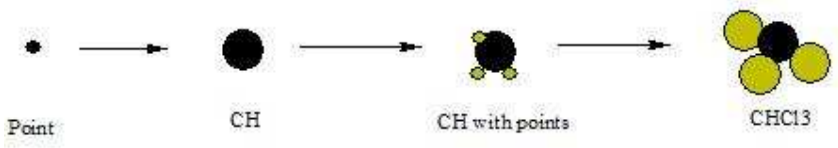


Fig. 3.5 Process for growing a chloroform molecule from a point particle.

3.3 Results and Discussion

3.3.1 Pure water in the CNT

3.3.1.1 Radial distribution functions

GCMC calculations were performed for water with a single (6, 6) CNT at a temperature of 298 K and a bulk water chemical potential of -11.61 kcal/mol. The simulations were performed for 5,000,000 cycles following equilibration. Under these conditions, water is present both inside and outside of the CNT, with the interior water consisting of a hydrogen bonding chain as shown in Fig. 3.6.1.

RDFs for water oxygen and hydrogen atoms in the CNT interior are displayed in Fig 3.6.2. As noted above, these functions are measured relative to the CNT axis, so the peaks at r = 0 correspond to atoms in the center of the CNT. Two oxygen peaks are evident in the plot, one at the CNT center and another displaced by 0.6 Å. The two peaks may be associated with water molecules in different hydrogen bonding configurations. The majority of the water molecules in the water chain donate one hydrogen bond and accept one hydrogen bond. These lead to the peak in $g_{O\text{-}CNT}(r)$ at 0.6 Å. Occasionally, however, water is observed to donate two hydrogen bonds or to accept two hydrogen bonds. These configurations correspond to more centralized oxygen atoms. There are also two distinct peaks in the RDF function for hydrogen, $g_{H\text{-}CNT}(r)$, as seen in Fig. 3.6.2. The central peak corresponds to hydrogen atoms that participate in hydrogen bonding while the outer peak represents hydrogen atoms pointing outward from the

hydrogen bonding chain. The ratio of atoms constituting these two peaks is close to 1:1 based on integration of the g(r) functions.

Formation of a linear hydrogen bonding chain from bulk water involves breaking of roughly 50% of water's hydrogen bonds. It is therefore surprising that water readily enters the hydrophobic CNT interior. Previous simulations of water inside CNTs have shown interior, linear-chain structures similar to those reported here[32,35]. The attractive interaction between water and the CNT plays a crucial role in formation of these interior structures.[35]

Water radial distribution functions on the outside of the CNT are shown in Fig. 3.6.3. The similar positions of the first peaks in the $g_{O\text{-}CNT}(r)$ and $g_{H\_CNT}(r)$ functions indicate that water forms a network of hydrogen bonds oriented tangentially to the CNT surface, a result consistent with water structures around small nonpolar solutes[56]. Structure beyond the first peak shows oscillations typical of fluid layering near any solid surface. Radial distribution functions for water outside of (6, 6) CNTs have not been reported previously in the literature. However, such structures around larger CNTs have been reported and appear similar to those shown here.[63,64]

3.3.1.2 Binding energy

Probability distributions for the binding energy *u*, as described in Sect. 3.2.2, are shown in Fig 3.7. The solid line represents the energy distribution function for bulk water under ambient conditions. Its values range between from

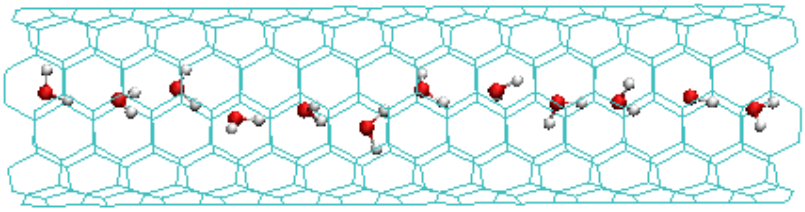


Fig. 3.6.1 Snapshot of a water configuration inside CNT.

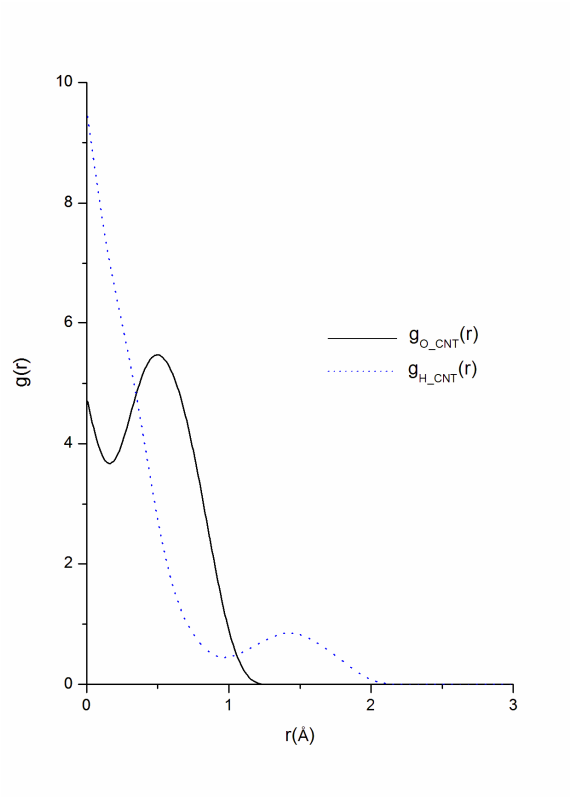


Fig.3.6.2 Radial distribution functions $g_{O\text{-}CNT}(r)$ (solid line) and $g_{H\text{-}CNT}(r)$ (dashed line) in the interior of a (6, 6) CNT.

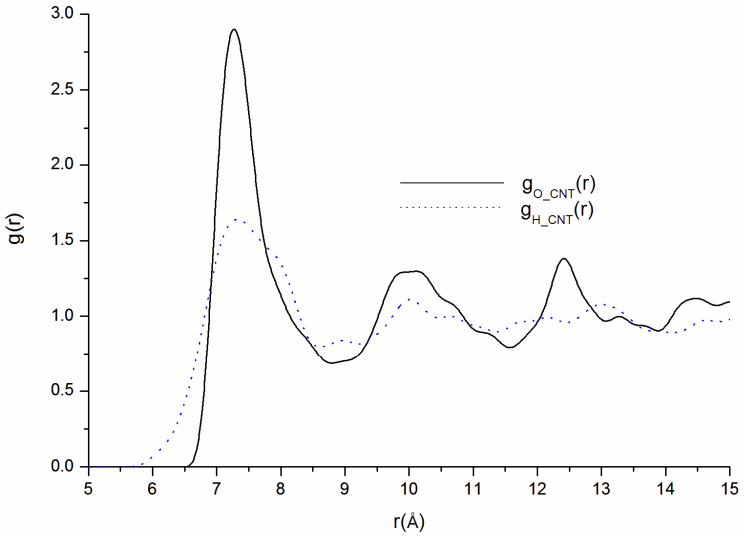


Fig. 3.6.3 Radial distribution functions $g_{O\text{-}CNT}(r)$ (solid line) and $g_{H\text{-}CNT}(r)$ (dashed line) outside of a (6, 6) CNT.

-35 kcal/mol and -10 kcal/mol with a maximum at an energy of -22.45 kcal/mol. The dashed line represents the binding energy distribution of water inside the (6, 6) CNT. It is narrower than the bulk water function and its peak is shifted to a higher energy of -18.05 kcal/mol. The narrow nature of the peak inside the CNT indicates that interior water molecules have a more uniform structure than those in bulk water. The shift in energy is consistent with a reduction by almost half in hydrogen bonds per water molecule, from almost 4 in bulk water to 2 in the CNT interior. However, the fact that the energy shift is much less than half of the bulk water value suggests that the hydrogen bonds formed in the CNT interior are very strong. Finally, results in Fig. 3.7 are in good agreement with similar calculations described in Ref. 35, providing a validation of the binding energy calculations as well as of the CNT simulations in general.

3.3.2 Hydrated Chloroform/CNT Systems

3.3.2.1 Chloroform radial distribution functions

RDFs for the chloroform C atom in hydrated CNT systems containing both 20 and 50 chloroform molecules are plotted in Fig. 3.8. Both curves show significant peaks at radial distances near 8 Å, indicating that the chloroform molecules bind preferentially over water to the CNT surface. Structures for the two curves beyond this first peak differ somewhat, but noise in the calculations due to the relatively small number of chloroform molecules makes interpretation of these features problematic. Representative snapshots of the system structure

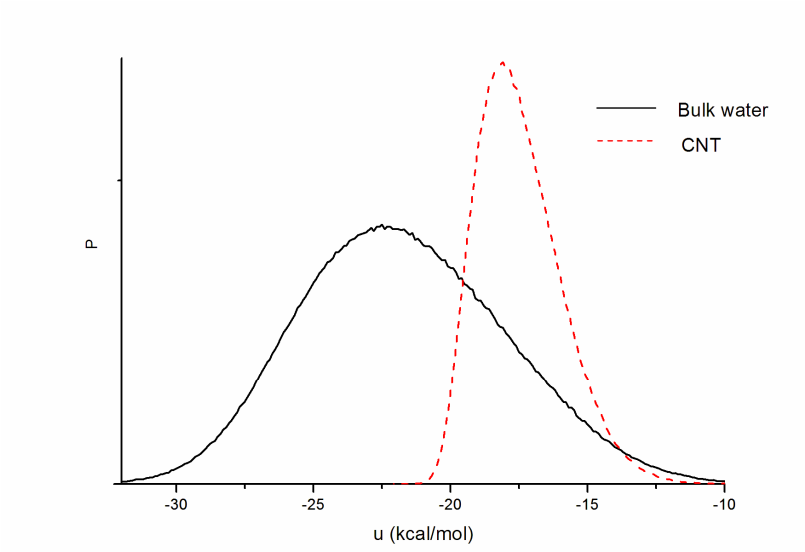


Fig. 3.7 Calculated binding energy distributions for water inside the CNT (dashed line) and in bulk water (solid line) at 298 K.

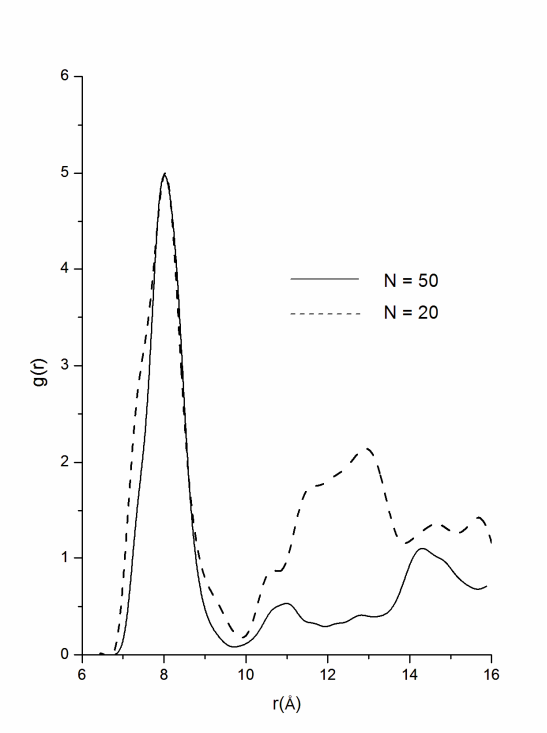


Fig. 3.8 Plots of $g_{C\text{-}CNT}(r)$ for systems containing 20 (dashed line) and 50 (solid line) chloroform molecules.

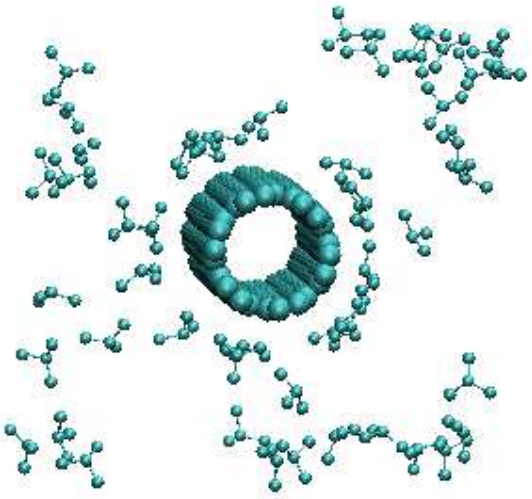


Fig. 3.9.1 Snapshot of 50 chloroforms in aqueous solution with CNT, water molecules are not shown on the picture.

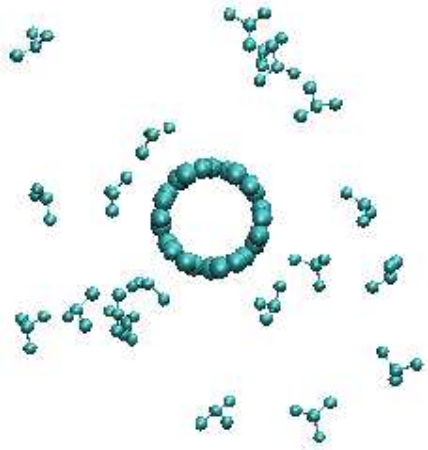


Fig. 3.9.2 Snapshot of 20 chloroforms in aqueous solution with CNT, water molecules are not shown on the picture.

for both N = 20 and N = 50 systems, displayed in Fig. 3.9, show clearly the preferentially binding of chloroform molecules to the CNT. Water molecules are omitted from these snapshots for clarity.

CNT-based adsorption materials would obviously consist of multiple CNT strands rather than the single strand considered here to this point. To begin an investigation of more complicated material configurations, we measured chloroform structure in the vicinity of two crossed CNT strands. A representative snapshot of this system containing 50 chloroform molecules is shown in Fig. 3.10.1. A two-dimensional density contour plot for chloroform molecules in this system is displayed in Fig 3.10.2. In this graph, the value of r_1 represents the distance from the axis of the CNT that is oriented in the x direction, while r_2 represents the distance from the axis of the CNT oriented in the z direction. The behavior of this contour plot is as expected given the RDFs observed around a single peak. Greater density for chloroform is observed in the contact region at around r = 8 Å along both CNTs, while the largest peak in the contour plot is near the position (r_1 = 8.0 Å, r_2 = 8.0 Å) around the joint of the two CNTs. Experimentally prepared CNT materials would certainly be much more complex than the simple crossed-CNT structure considered here, so further studies of how CNT-CNT structure may have a cooperative effect on chloroform adsorption are still warranted.

CNT-based adsorption materials would obviously consist of multiple CNT strands rather than the single strand considered here to this point. To begin an

investigation of more complicated material configurations, we measured chloroform structure in the vicinity of two crossed CNT strands. A representative snapshot of this system containing 50 chloroform molecules is shown in Fig. 3.10.1. A two-dimensional density contour plot for chloroform molecules in this system is displayed in Fig 3.10.2. In this graph, the value of r_1 represents the distance from the axis of the CNT that is oriented in the x direction, while r_2 represents the distance from the axis of the CNT oriented in the z direction. The behavior of this contour plot is as expected given the RDFs observed around a single peak. Greater density for chloroform is observed in the contact region at around r = 8 Å along both CNTs, while the largest peak in the contour plot is near the position (r_1 = 8.0 Å, r_2 = 8.0 Å) around the joint of the two CNTs. Experimentally prepared CNT materials would certainly be much more complex than the simple crossed-CNT structure considered here, so further studies of how CNT-CNT structure may have a cooperative effect on chloroform adsorption are still warranted.

3.3.2.2 Binding Free Energy

Binding free energy profiles for chloroform in the systems with N = 20 and N = 50 chloroform molecules were determined from their respective g(r) functions by inverting Equation 3.2. Results are displayed in Fig 3.11. The shapes of the w(r) curves are nearly identical inside of 10 Å for the two systems, but differ outside of this region. These results suggest that binding of chloroform to the surface of the CNT is strong and relatively insensitive to the local solvation

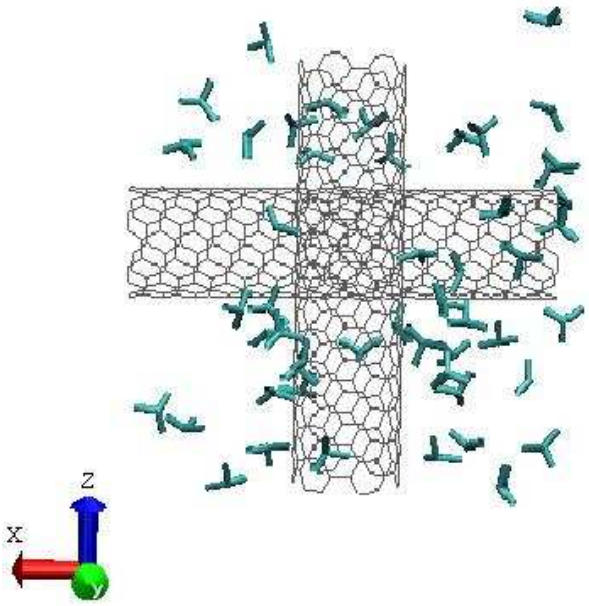


Fig. 3.10.1 Snapshot of the hydrated system containing two crossed CNTs and 50 chloroform molecules. Water molecules are not shown.

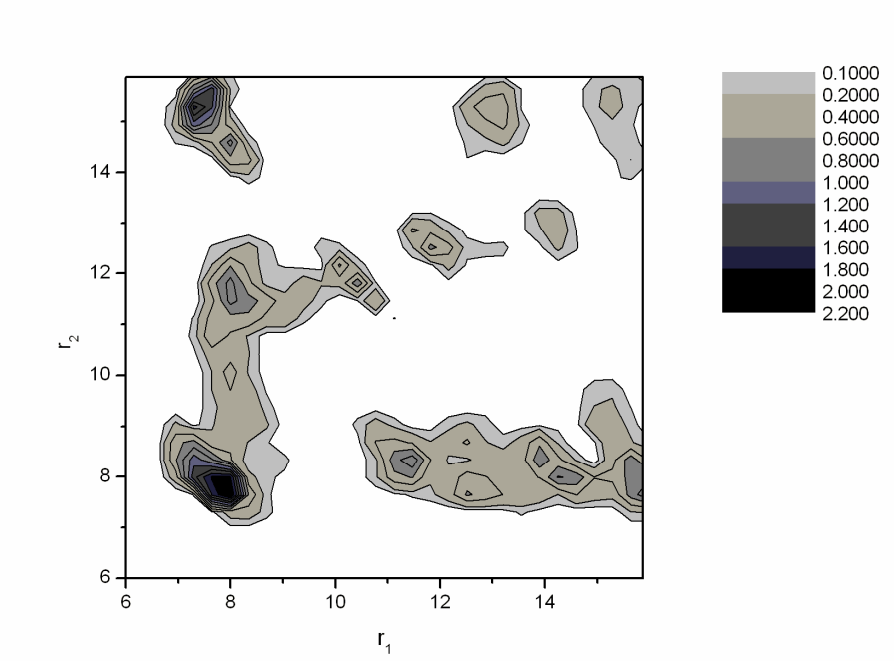


Fig. 3.10.2 Contour plot of the chloroform distribution in the crossed CNT system. The value of r_1 represents the distance from the CNT oriented along the x axis, while r_2 represents the distance from the CNT oriented along the z axis.

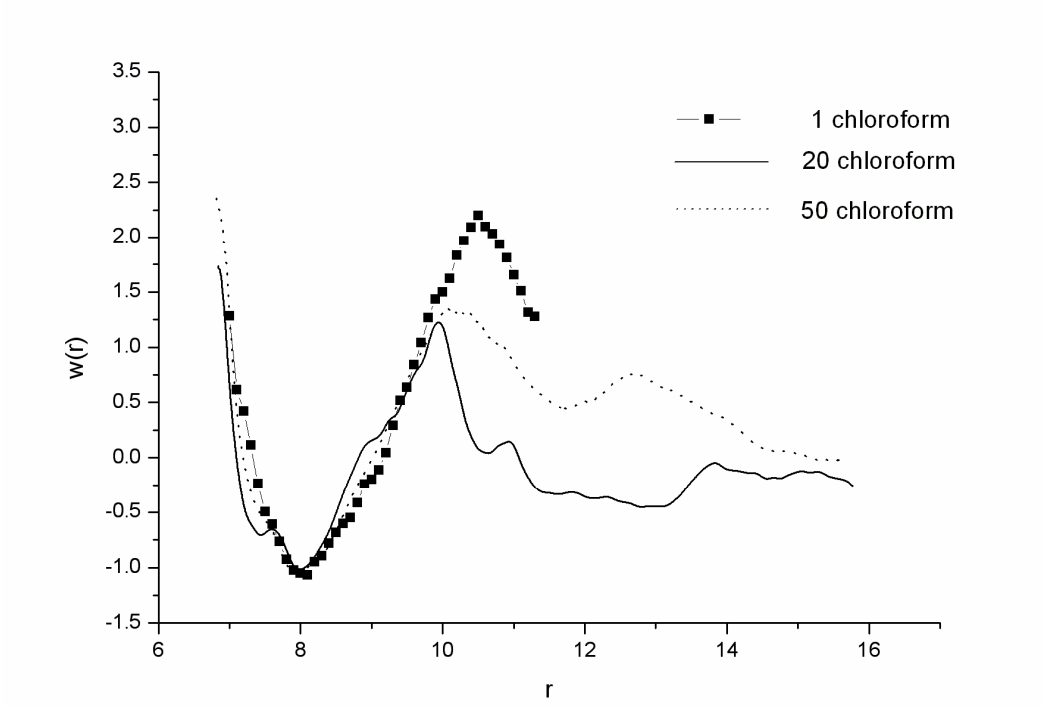


Fig. 3.11 Binding free energies for chloroform as a function of radial distance from the CNT axis for systems with N = 1 (square symbols), N = 20 (solid line) and N = 50 (dashed line) chloroform molecules. Results for N = 20 and N = 50 are determined from g(r) functions using Eq. 3.2. Results for N = 1 are from a thermodynamic integration calculation.

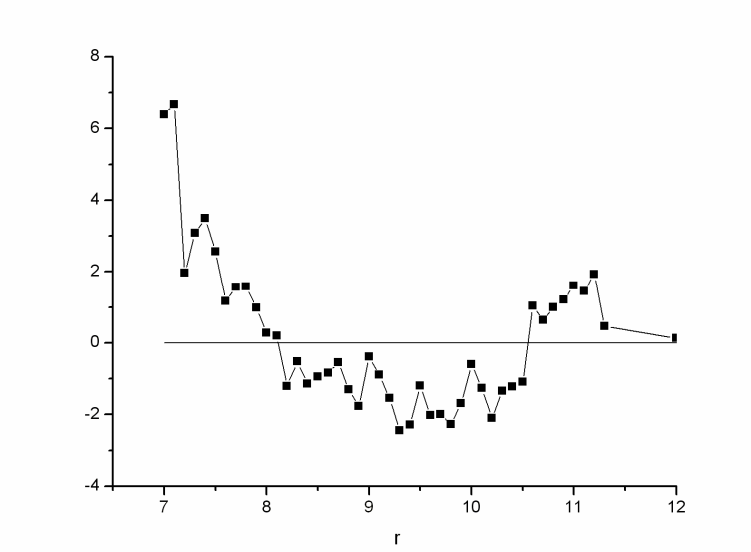


Fig. 3.12 Average radial force on a chloroform molecule as a function of its distance from the CNT axis.

with evidence of a "second shell" adsorption layer for N = 20 but not N = 50. Signal to noise limitations prevent a complete analysis of these data.

To further assess the chloroform concentration dependence of binding, the chloroform binding free energy profile was determined for a single chloroform molecule using the thermodynamic integration method[61] as described in Section 3.2.6. The average radial force on a constrained chloroform molecule calculated as a function of distance from the CNT axis is shown in Fig.3.12. The measured force is positive (i.e., repulsive) inside of a distance of 8 Å from the CNT axis, negative for distances between around 8 and 10.5 Å, and positive again outside of 10.5 Å.

Integration of the average force in Fig. 3.12 according to Eq. 3.3 produces the binding free energy (i.e., the PMF) that is displayed as square symbols in Fig. 3.11. The shape of the binding free energy curves for the three systems shown in that figure are remarkably similar in the region inside of 10 Å. This suggests that the binding free energy is remarkably insensitive to the local solvation environment, that is, whether the local environment consists entirely of water molecules (as for the N = 1 case) or predominantly chloroform molecules (as for N = 50). The curves deviate outside of 10 Å, with the barrier to chloroform-CNT dissociation being largest for the N = 1 case.

3.3.2.3 Orientation

The orientation of chloroform at different distances from the CNT axis may

be described in terms of a cos(θ) orientational distribution function (ODF), where θ is defined as in Fig. 3.4. The ODFs were determined simultaneously with the N = 1 average force calculations, at distances from the CNT axis ranging between 7.0 to 12.0 Å. Results of the ODF calculations at several chloroform-CNT distances are displayed in Figs.3.13.1-3.13.4. For chloroform-CNT distances of less than around 7.7 Å, there are two stable orientational states for chloroform that do not interchange on the simulation timescale. Measured ODFs therefore depend on the starting orientation in these cases. In one of these configurations, the chlorine atoms are all pointing inward ($\cos(\theta) \approx -1$), while in the other they all point outward ($\cos(\theta) \approx +1$). Analysis of orientational structures observed for the N = 50 system, discussed below, suggest that only the "outward" configurations are accessible at room temperature. Therefore, only ODFs generated from outward starting configurations are shown in the figure.

The first plot, in Fig. 3.13.1, shows the ODF generated with an outward-pointing starting configuration in which the CNT-chloroform distance was constrained to be 7.5 Å. The distribution shows a single peak, centered at $\cos(\theta) \sim 0.7$. Clearly the chloroform molecule always points in an outward direction since the value of cos(θ) is never less than zero. However, the favored orientation does not point directly out ($\cos(\theta) = 1$) but rather is tipped somewhat, probably to optimize the CNT-Cl distance for one or two of the chloroform chlorine atoms. As the CNT-chloroform distance is increased to 8.0 Å (Fig. 3.13.2), the

outward-pointing peak shifts to smaller values of cos(θ), now centered near a value of 0.2 and including population for both positive and negative values of cos(θ). In addition, a sharp second peak at an inward-pointing value of cos(θ) = -1.0 is now evident, indicating the molecule can flip between inward and outward configurations. Two peaks are still evident when the CNT-chloroform distance is increased to 9.0 Å (Fig. 3.13.3). The outward peak is now centered at around cos(θ) = 0.3 while the inward peak has broadened significantly and shifted away from cos(θ) = -1.0 to near -0.6. Finally, Fig. 3.13.4 shows the OCF for a CNT-chloroform distance of 10.5 Å, near the barrier to chloroform dissociation shown in Fig. 3.11. The distribution now consists of a single, broad inward-pointing peak centered at cos(θ) = -0.6. This suggests that a significant portion of the driving force for CNT-chloroform binding is the strong Cl-CNT attraction.

Orientational distributions for the system containing 50 chloroform molecules were also calculated and are plotted in Fig 3.14 as a contour plot. Unlike in the N = 1 ODFs, chloroform molecules were not constrained during the calculation of this contour plot. Three primary peaks are evident in the figure. The left-most peak centered near r = 8.2 Å and cos(θ) = -1.0 has the largest population density. This inward-pointing configuration corresponds to the sharp inner peak in Fig. 3.13.2. The second peak centered near r = 8.5 Å and cos(θ) = -0.5 corresponds to a convergence of the two peaks seen in Fig. 3.13.2 . The third peak centered

near r = 7.6 Å and cos(θ) = +0.4 represents favorable outward-pointing configurations corresponding to structures observed in Fig. 3.13.1. In general, the chloroform structures for the N = 1 and N = 50 systems are very similar inside of r = 8.5 Å. Structures outside of 8.5 Å, including in the barrier region, for the N = 50 system were difficult to determine, however, due to relatively small population of chloroform molecules in these regions.

3.3.2.4 Solvation

The Thermodynamic Integration (TI) method[61] was used to determine Helmholtz free energy of solvation, ΔA, for a chloroform molecule associated with the CNT. The chloroform molecule was created in a two-stage process as described in Section 3.2.8. The first stage involves creating an argon atom from a non-interacting point particle, while the second stage involves converting the argon atom into a chloroform molecule. The particle is constrained to be a distance of 7.0 Å from the CNT for both stages. Orientational motion is also constrained during the second stage with the molecule created in an parallel (cos(θ) = 0) configuration. Results are given here for each stage separately.

3.3.2.4.1 Solvation free energy of argon

Results for the first stage, creating an argon atom from an ideal-gas particle, are given in Figs. 3.15.1 and 3.15.2. The first of these figures shows a plot of the force along the λ-coordinate, $<\partial U / \partial \lambda>$, for values of λ ranging from 0 to 1 with a step size of 0.1. The measured force is positive except in the range from λ =

0.4 - 0.5, and fluctuates significantly, probably due to a poor signal to noise ratio. Convergence is slow for these calculations due to the relatively large system size. Fig. 3.15.2 gives the Helmholtz free energy of solvation determined by integration of the force in Fig. 3.15.1. The final value of ΔA = 1.8 kcal/mol is similar to the value of 2.3 kcal/mol determined for argon in bulk water, described above.

3.3.2.4.2 Solvation free energy of chloroform

Results for the second stage, in which a chloroform molecule is grown from the argon atom that was created in the first stage, are shown in Figs. 3.16.1 and 3.16.2. The measured force along the λ coordinate (Fig. 3.16.1) is strongly positive for larger values of λ due to the repulsive interaction between the chlorine atoms and the CNT. This leads to a large, positive solvation free energy (Fig. 3.16.2). The total value of *A* for the second stage is equal to 12.2 kcal/mol. When combined with the results for the first stage, this gives a total solvation free energy of 14.0 kcal/mol.

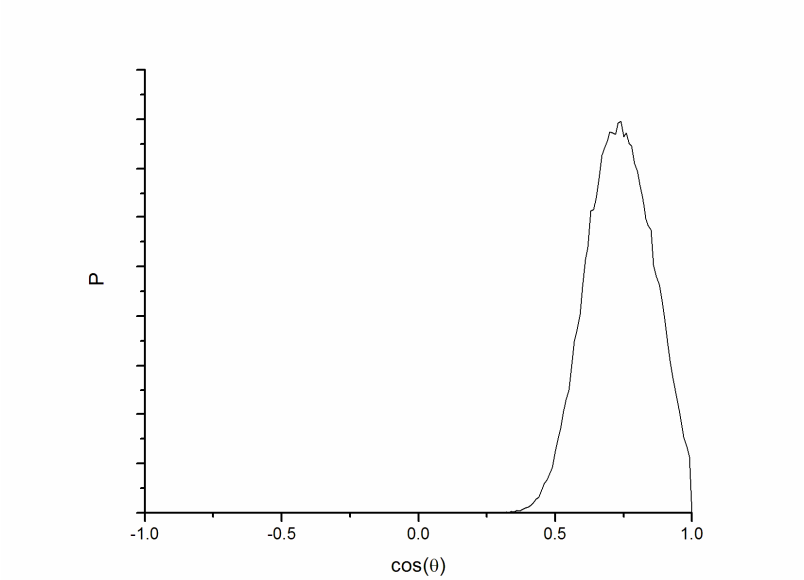


Fig. 3.13.1 Orientational distribution function for chloroform at a distance of 7.5 Å from the CNT axis.

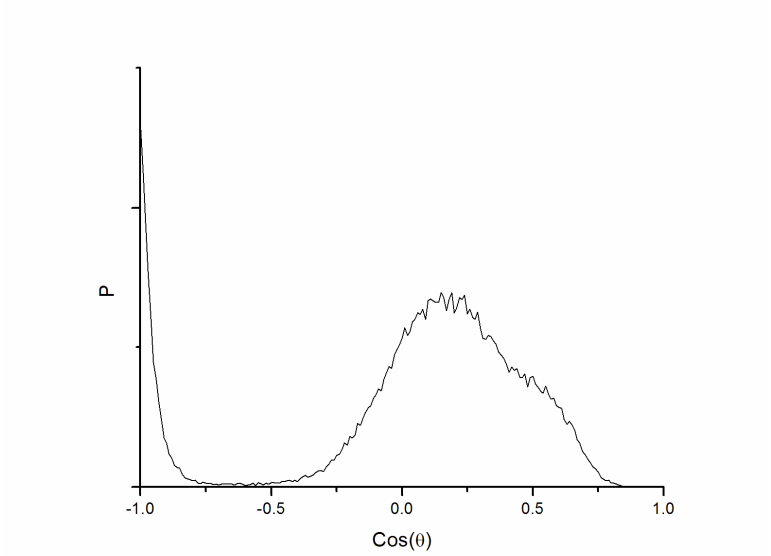


Fig. 3.13.2 Orientational distribution function for chloroform at a distance of 8.0 Å from the CNT axis.

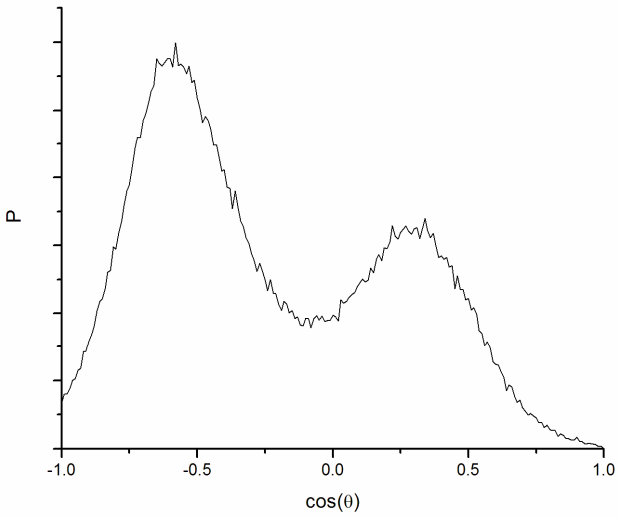


Fig. 3.13.3 Orientational distribution function for chloroform at a distance of 9.0 Å from the CNT axis.

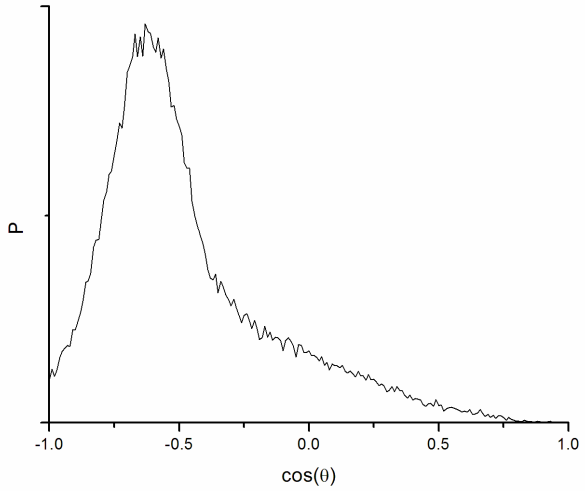


Fig. 3.13.4 Orientational distribution function for chloroform at a distance of 10.5 Å from the CNT axis.

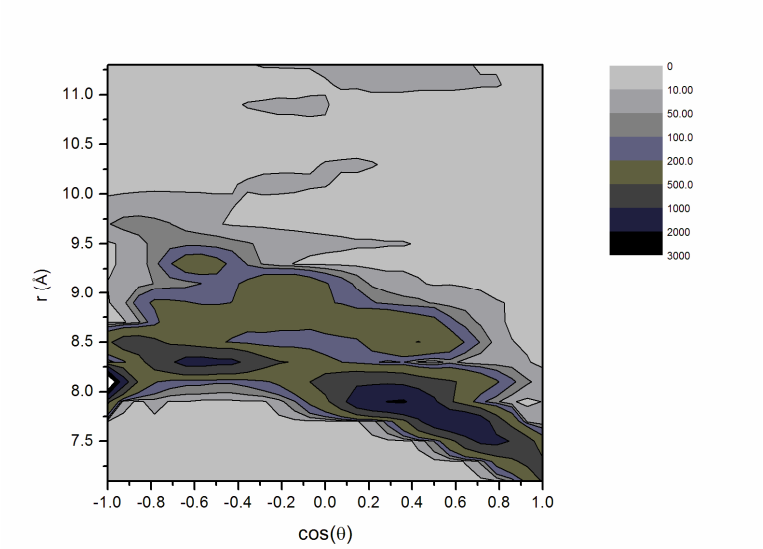


Fig. 3.14 Contour plot of the orientational distribution functions for chloroform as a function of distance from the CNT axis.

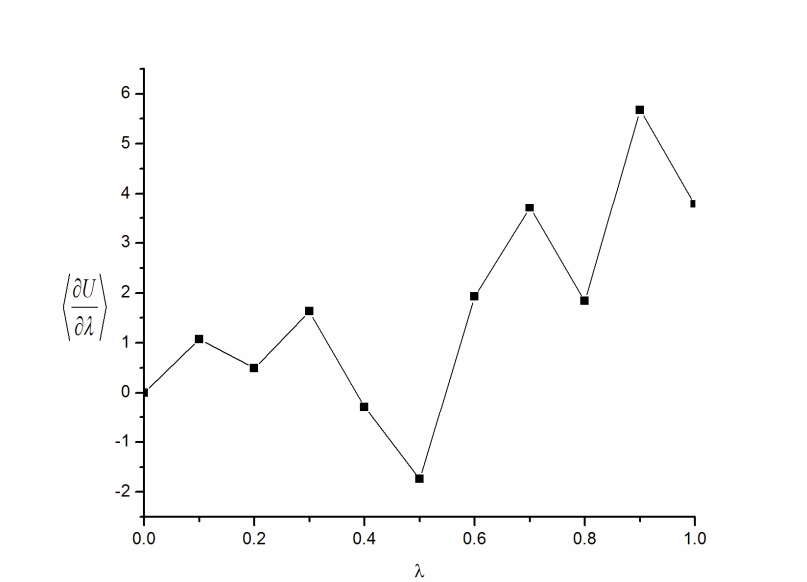


Fig. 3.15.1 Average force along the λ coordinate for a particle constrained to be a distance of 7.0 Å from the CNT axis. ΔA value of λ=0 corresponds to a non-interacting (i.e. ideal gas) atom, while λ=1 corresponds to an argon atom.

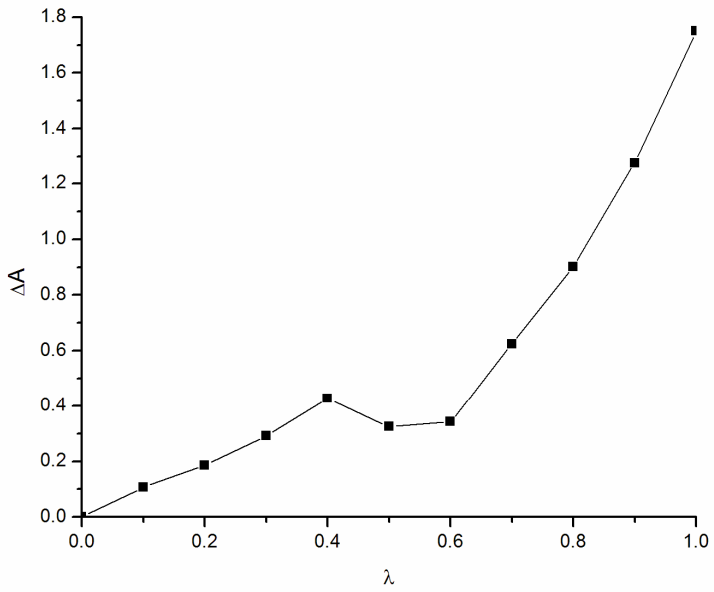


Fig. 3.15.2 Helmholtz free energy of solvation, ΔA, plotted as a function of the coupling parameter λ as determined by integration of the data in Fig. 3.15.1. The value of the function at λ=1 is the free energy of solvation of an argon atom constrained at a distance of 7.0 Å from the CNT axis.

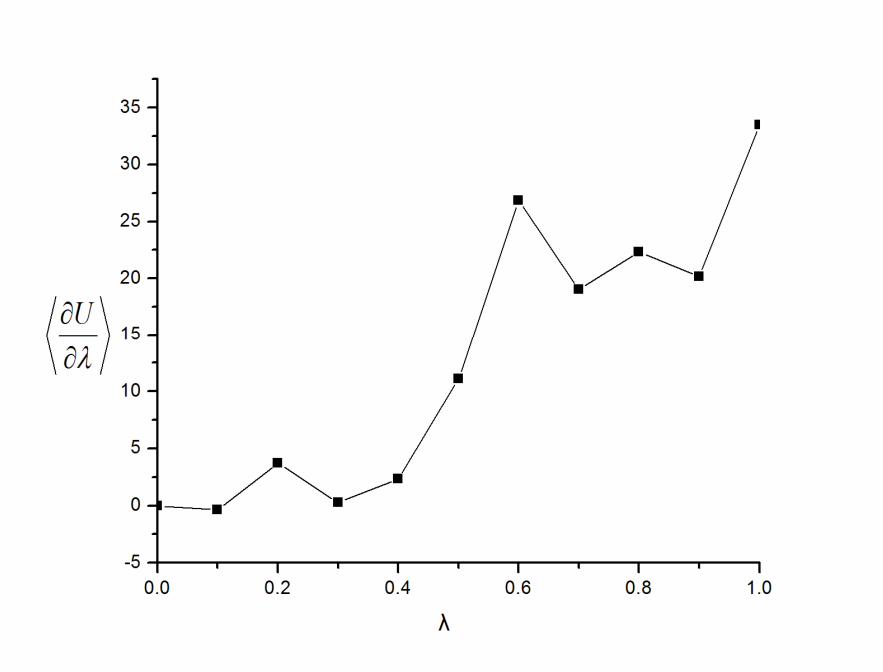


Fig. 3.16.1 Average force along the λ coordinate for a particle constrained to be a distance of 7.0 Å from the CNT axis. ΔA value of λ=0 corresponds to an argon atom, while λ=1 corresponds to a chloroform molecule.

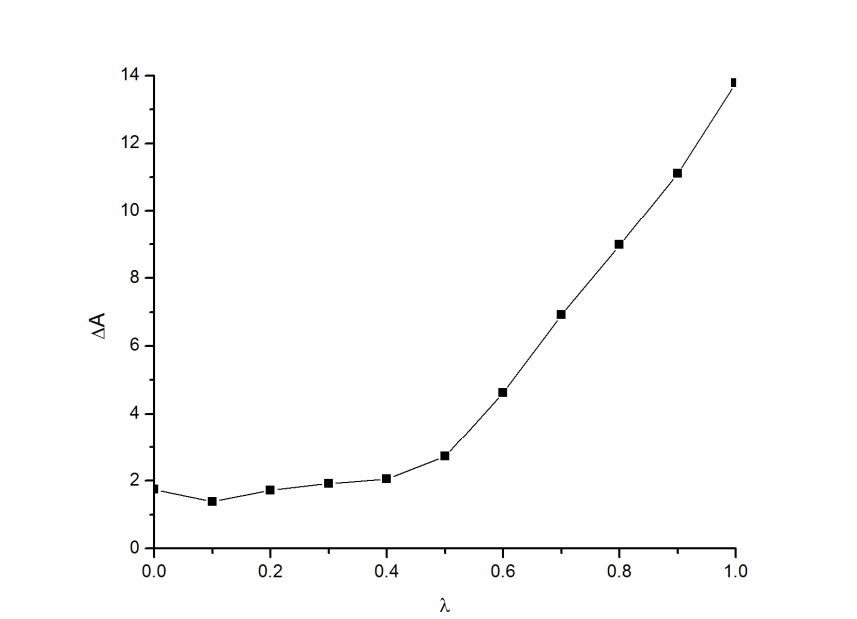


Fig.3.16.2 Helmholtz free energy of solvation, ΔA, plotted as a function of the coupling parameter λ as determined by integration of the data in Fig. 3.16.1. The value at is λ=0 taken from the end point of Fig. 3.15.2 and corresponds to the free energy of solvation of an argon atom. The value of the function at λ=1 is the free energy of solvation of a chloroform molecule constrained at a distance of 7.0 Å from the CNT axis.

3.4 References

[1] Iijima, S. *Nature 1991, 354*, 56.

[2] Radushkevich, L.; Lukyanovich, V. *Zh. Fis. Khim. 1952, 26*, 88.

[3] Oberlin, A.; M. Endo, and T. Koyama, J. Cryst. Growth (March 1976). "Filamentous growth of carbon through benzene decomposition". *Journal of Crystal Growth* 32: 335–349., 603.

[4] US patent 4663230, *"Carbon fibrils, method for producing same and compositions containing same"*, granted 1987-05-05.

[5] Iijima, S.; Ichihashi, T. *Nature 1993, 363*, 603.

[6] Bethune, D.S.; Kiang, C.H.; Devries, M.S.; Gorman, G.; Savoy, R.; Vazquez, J.; Beyers, R. *Nature 1993, 363*, 605.

[7] Ajayan, P.M. *Chem. Rev. 1999, 99*, 1787.

[8] Dresselhaus, M.; Dresselhaus, G.; Eklund, P.; Saito, R. *Phys. Would 1998, 11*, 33.

[9] Alexiadis, A.; Kassinos, S. *Chem. Rev. 2008, 108*, 5014.

[10] Cheng, H.M. ; Yang, Q.H. ; Liu, C. *Carbon 2001, 39*, 1447.

[11] Zuttel, A. ; Sudan, P. ; Mauron, P. ; Kiyobayashi, T. ; Emmenegger, C. ; Schlapbach , L. *Int J. Hydrogen Energy 2002, 27*, 203.

[12] Cao, D.P. ; Zhang, X.R. ; Chen, J.F. ; Wang, W.C. ; Yun, J. *J. Phys. Chem. B 2003, 107*, 13286.

[13] Liu, C. ; Fan, Y.Y. ; Liu, M. ; Cong, H.T. ; Cheng, H.M. Dresselhaus, M.S. *Science 1999, 286*, 1127.

[14] Baughman, R.H. ; Cui, C.X. ; Zakhidov, A.A. ; Iqbal, Z. ; Barisci, J.N. ; Spinks, G.M. ; Wallace, G.G. ; Mazzoldi, A. ; De Rossi, D. ; Rinzler, A.G. ; Jaschinski, O. ; Roth, S. ; Kertesz, M. *Science 1999, 284*, 1340.

[15] Sazonova, V. ; Yaish, Y. ; Ustunel, H. ; Roundy, D. ; Arias, T.A. ; McEuen, P.L. *Nature 2004, 431*, 284.

[16] Kong, J. ; Franklin, N.R. ; Zhou, C.W. ; Chapline, M.G. ; Peng, S. ; Cho, K.J. ; Dai, H.J. *Science 2000, 287*, 622.

[17] Snow, E.S. ; Perkins, F.K. ; Houser, E.J. ; Badescu, S.C. ; Reinecke, T.L. *Science 2005, 307*, 1942.

[18] Hinds, B.J. ; Chopra, N. ; Rantell, T. ; Rews, R. ; Gavalas, V. ; Bachas, L.G. *Science 2004, 303*, 62.

[19] Kalra, A. ; Garde, S. ; Hummer, G. *Proc. Natl. Acad. Sci, U.S.A. 2003, 100,* 10175.

[20] Hong, M.H. ; Kim, K.H. ; Bae, J. ; Jhe, W. *Appl. Phys. Lett. 2000, 77*, 2604.

[21] Kim, P. ; Lieber, C.M. *Science 1999, 286*, 2148.

[22] Baughman, R.H. ; Zakhidov, A.A. ; de Heer, W.A. *Science 2002, 297*, 787.

[23] Power, T.D. ; Skoulidas, A.I. ; Sholl, D.S. *J. Am. Chem. Soc. 2002, 124*, 1858.

[24] Portney, N.G. ; Ozkan, M. *Anal. Bioanal. Chem. 2006, 384*, 620.

[25] Singh, R. ; Pantarotto, D. ; Lacerda, L. ; Pastorin, G. ; Klumpp, C. ; Prato, M. ; Bianco, A. ; Kostarelos, K. *Proc. Natl. Acad. Sci. U.S.A. 2006, 103*, 3357.

[26] Cheng, H.S. ; Cooper, A. C. ; Pez, G.P. ; Kostov, M.K. Piotrowski, P. ; Stuart, S.J. *J. Phys. Chem. B 2005, 109*, 3780.

[27] Sansom, M. S. P. ; Kerr, I.D. ; Breed, J. ; Sankararamakrishnan, R. *Biophys. J. 1996, 70*, 693.

[28] Sansom, M.S.P. ; Biggin, P.C. *Nature 2001, 414*, 156.

[29] Floquet, N. ; Coulomb, J.P. ; Dufau, N. ; Re, G. *J. Phys. Chem. B 2004, 108*, 13107.

[30] A. Striolo, A.A. Chialvo, K.E. Gubbins, P.T. Cummings. *The Journal of Chemical Physics 2005, 122*, 234712.

[31] Christoph Dellago, Mor M. Naor, and Gerhard Hummer. *Plhysical Review Letters 2003, 90,* 105902-1.

[32] A. Striolo, K.E. Gubbins, M.S. Gruszkiewicz, D.R. Cole, J.M. Simonson, A.A. Chialvo, P.T. Cummings, T.D. Burchell, K.L. More. *Langmuir 2005, 21,* 9457.

[33] Gordillo, M.C.; Marti, J. Chem. *Phys. Lett. 2000, 329,* 341.

[34] Alexiadis, A.; Kassinos, S. *Chem. Rev. 2008, 108,* 5014.

[35] G. Hummer, J.C. Rasaihah & J.p. Noworyta. *Nature 2001, 414,* 188.

[36] Liu, Y.C.; Wang, Q.; Wu, T.; Zhang, L. *J. Chem. Phys. 2005, 123,* 234701.

[37] Waghe, A.; Rasaiah, J.C.; Hummer, G. *J. Chem. Phys. 2002, 117,* 10789.

[38] Kolesnikov, A.I.; Loong, C.K.; de Souza, N.R. Burnham, C.J.; Moravsky, A.P. *Phys. B. Condens. Matter 2006, 385,* 272.

[39] Kolesnikov, A.I.; Zanotti, J.M.; Loong, C.K. Thiyagarajan, P.; Moravsky, A.P.; Loutfy, R.O.; Burnham, C.J. *Phys. Rev. Lett. 2004, 93,* 035503.

[40] de Souza, N.R.; Kolesnikov, A.I.; Burnham, C.J.; Loong, C.K. *J. Phys.: Conens. Matter 2006, 18,* 2321.

[41] Zheng, J.; Lennon, E.M.; Tsao, H.K.; Sheng, Y.J.; Jiang, S.Y. *J. Chem. Phys. 2005, 122,* 214702.

[42] Joseph, S.; Mashl, R.J.; Jakobsson, E.; Aluru, N.R. *Nano Lett. 2003, 3,* 1399.

[43] Huang, B.D.; Xia, Y.Y.; Zhao, M.W.; Li, F.; Liu, X.D.; Ji, Y.J.; Song, C. *J. Chem. Phys. 2005, 122,* 084708.

[44] Vaitheeswaran, S.; Rasaiah, J.C.; Hummer, G. *J. Chem. Phys. 2004, 121,* 7955.

[45] Valentini, L.; Armentano, I. Kenny, J.M. *Diamond Relat. Mater. 2005, 14,* 121.

[46] http://en.wikipedia.org/wiki/Chloroform, visited at May 16, 2009.

[47] Dumas, J.B. *Annalen der Pharmacie 1834, 107,* 650.

[48] www.atsdr.cdc.gov/toxpro2.html, visited at May 16, 2009.

[49] Agency for Toxic Substances and Disease Registry (ATSDR). *1997*. Toxicological profile for chloroform. Atlanta, GA: U.S. Department of Health and Human Services, Public Health Service

[50] http://www.npi.gov.au/database/substance-info/profiles/23.html#health, visited at May 16, 2009.

[51] Lu, A.H.; Guo, Y.J.; Liu, J.; Liu, F.; Wang, C.Q.; Li, N.; Li, Q. *Chinese Science Bulletin 2004, 49*, 2350.

[52] Peng, Q.T; Hou, X.J.; Cui, L. *Medical Journal of Gentral Equipment 2005, 7*, 783.

[53] Tao, Y.Y.; Kang, J.; Xu, Z.; Zhao, Y. *Huanjing Gongcheng 2007, 25*, 44.

[54] Ramsahye, N.A.; Bell, R.G. *J. Phys. Chem. B 2005, 109*, 4738.

[55] Krasnova, T.A; Kirsanov, M.P.; Ushakova, O.I. *Zhurnal Fizicheskoi Khimii 2001, 75*, 1912.

[56] Barkaline, V.V.; Chashynshi, A.S. *Rev. Adv. Mater. Sci. 2009, 20*, 21.

[57] Diaz, E.; Ordonez, S.; Vega, A. *J. Phys.: Conference Series 2007, 61*, 904.

[58] http://en.wikipedia.org/wiki/Carbon_nanotube, visited at May 16, 2009.

[59] Abrahamson, J.; Wiles, P.G. *Carbon 1999, 37*, 1873.

[60] Chandler, D. *Introduction to Modern Statistical Mechanics*; Oxford University press: *1987*.

[61] Mezei, M.; Beveridge, D.L. *Ann. N. Y. Acad. Sci. 1986, 482*, 1.

[62] Frenkel, D.; Smit, B. *Understanding Molecular Simulation from Algorithms to Applications*. *2002*, Academic press: London.

[63] Liu, Y.C.; Wang, Q.; Zhang, L.; Wu, T. *Langmuir 2005, 21*, 12025.

[64] Hanasaki, I.; Nakatani, A. *J. Chem. Phys. 2006, 124*, 144708.

Chapter 4

Methane storage in hydrated slit-pore systems

4.1 Background

Natural gas is one important component of a comprehensive energy policy that focuses on both meeting increasing energy demands and on mitigating global warming and climate change effects[1]. Environmental needs motivate a long-term reduction in the dependence on carbon-based fuel sources. For example, an agreement by industrialized countries in 1987 requires reduction of greenhouse gas emission to 5.2% below 1990 levels by 2010[2]. Nevertheless, meeting short-term energy demands will likely require significant use of natural gas due to its relative abundance among remaining fossil-fuel resources. In addition, natural gas can be generated from plant matter and hence may become an important renewable energy resource[3,4].

One important focus of an energy policy is portable fuel sources for use in automobiles. Hydrogen and natural gas fuel cells are an important enabling technology and have the potential to revolutionize the way we power our nation. They are energy-efficient, clean, portable and fuel-flexible. In addition, over 20% to 40% energy service costs will be saved compared to conventional combustion of natural gas[5]. The U.S. Department of Energy (DOE) has funded projects for development of fuel cells[6], and portable fuel cells have already reached the commercial market in some niche applications. One major challenge to the

practical use of fuel cells in automotive applications is fuel storage. Hydrogen gas storage technologies are still at a primitive level. In contrast, recent breakthroughs in natural gas storage methods suggest that use of natural-gas fuel cells may become feasible on a more reasonable time scale.

There are two parameters usually associated with fuel sources that characterize their usefulness: heat value and energy density. A high heat value implies a relative small fuel mass requirement, while a high energy density is associated with small storage volume requirements. Natural gas has a reasonable heat value. However, since it is gaseous under ambient conditions, it has a very small energy density, about $1/1000^{th}$ that of gasoline[7] and cannot be used directly under such conditions in an automotive application. Three methods have been proposed for increasing energy density of natural gas: compressed natural gas (CNG), liquefied natural gas (LNG) and adsorbed natural gas (ANG). The first two are commercially available techniques. CNG has an operational pressure between 163 and 296 atm and therefore is associated with safety concerns while requiring a highly pressure resistant container. Such storage containers are very heavy and also require an expensive compressor for filling. LNG provides the maximum volumetric energy density for natural gas. However, liquefying natural gas cannot be done without cooling to below methane's critical temperature (191 K) which is a costly and energy consuming process. The ANG method has the obvious virtue of a low storage pressure but, while promising in many respects[8], it

has yet to achieve desired energy density levels.

To obtain a suitable driving range for automotive applications, the commonly accepted target for natural gas storage density is 150 V/V at 3.45 atm and room temperature. The V/V unit here refers to the volume of natural gas under ambient conditions divided by the volume of adsorbent material. Various potential adsorbent materials have been investigated in this respect including superactivated carbon[9], pillared carbon clay[10], activated carbon fibers[11,12], carbon monoliths[13], single-walled carbon nanotubes and carbon nanohorns[14,15], yet none of these materials exhibited adequate V/V ratios. Recently, a promising new approach to natural gas adsorption involving *pre-hydrated* carbonaceous materials was introduced, through two Japanese patent applicaitons[16,17], in which V/V ratios of nearly 300 were claimed to have been obtained. A publication in 1998 by Miyawaki also reported that the pre-adsorbed water can enhance methane adsorption at 303K[18]. Subsequently, Zhou and coworkers reported dramatically enhanced methane adsorption due to pre-hydration of activated carbonaceous materials[9]. Following this initial work, several additional publications from two independent research groups established conclusively that hydrated carbonaceous materials show enhanced methane adsorption[19-26]. This enhanced adsorption has been attributed by both of these research groups to the formation of methane clathrate hydrate structures. Evidence for this includes that enhanced adsorption occurs only for pressures above the formation pressure (P_f) of hydrates, while

pre-hydration has instead a detrimental effect for adsorption pressures below P_f[26]. One potential drawback to this adsorption strategy is that temperatures may need to be constrained to the range from 273-283K[27] which corresponds to temperatures of stable methane hydrates in the bulk state. However, Miyawaki's results obtained at T = 303 K[18] suggest that this temperature range may be somewhat larger due to the constrained environment. Finally, the high adsorption capacity can be observed from macropoes, mesopores, and micropores even though the latter would seem to be unsuitably small for the formation of methane hydrates[21].

Despite some evidence that enhanced methane adsorption by pre-hydrated carbonaceous materials may be related to clathrate hydrate formation, a great deal of research is still required to characterize and understand the processes underlying this phenomenon. Computer simulations provide one route by which the adsorption mechanism in small pores may be investigated. However, the theoretical understanding of methane adsorption on wet carbonaceous materials is currently rudimentary. Muller et al.[28] simulated methane adsorption on activated carbon. They found that adsorption strongly depends on oxygenated site density, with increasing density reducing the required adsorption pressure to achieve a particular methane loading. The mechanism for loading also changed with increasing O-side density, switching from capillary condensation to a continuous pore filling process. Contrary to the experimental measurements discussed above, they found that methane adsorption always decreased with increased water

adsorption. The water molecules clustered around oxygenated sites and blocked a significant fraction of the surface for methane adsorption. Miyawaki[18] et al also performed simulations to compare with their experimental observations. They observed from their experiments that the water/methane ratio in a slit pore is lower than in a pure hydrate, causing them to speculate that a new type of clathrate is formed in slit pores. High temperature can improve the adsorption within 293-313K. They explain that the mobility of water plays an important role. But they reported the pressure of hydrate formation in slit pore is lower than pure hydrate. This is in agreement with with Zhou's report. In Sizov's[29] study, a cluster of five water molecules is surrounded by 22 methane molecules at 298K. Individual water molecules can not be retained in a pore and a cluster smaller than five molecules is not stable. They explained these phenomena as being due to the stronger interaction between water and methane than between water and the wall which contributes up to 10-12 kJ/mol to the stabilization energy of a microcluster. Aftermany simulation stpes, no methane was retained in the pore.

All these simulations are different from the experiments in which the water/carbon ratio is constant. In above simulations, the fluctuation of water amount in slit pore can repel methane due to strong hydrogen bonds in water. This does not occur in the experiments. In addition, their water/methane model is totally different from the clathrate structure in pure hydrates which is assumed to be formed in experiment. These divergences of understanding require more research.

The simulations described in this chapter constitute preliminary investigations of methane adsorption in hydrated slit pores. Simulations completed to this point are used first for code validation through reproduction of previously published quantities and second to establish some baseline behavior for methane and water in slit pore systems. These involve primarily measurement of structural quantities for both methane and water adsorption. The chapter ends with suggestions for future work that could help clarify how methane hydrate formation may be impacting methane adsorption in hydrated slit pores.

4.2 Method and theory

The system for the simulations described in this chapter consists of two walls representing collections of graphene layers divided by a single slit pore of variable thickness. It is designed to mimic the system in the experiments of Zhou and co-workers[21]. The walls are oriented perpendicular to the z axis with periodic boundary conditions applied in the x and y directions. The methane and water contents for the system will in some cases be held constant and in others regulated by controlling the chemical potential and using the GCMC simulation code.

4.2.1 Model

The interaction potential model used in these simulations is closely related to those used previously in studies of methane in graphite slit pores[30-32]. In this model, the interaction of a fluid atom (water or methane) with the solid graphene lattice is represented by the following simplified relationship

$$\phi_{sf}(z) = A\left[\frac{2}{5}\left(\frac{\sigma_{sf}}{z}\right)^{10} - \left(\frac{\sigma_{sf}}{z}\right)^{4} - \left(\frac{\sigma_{sf}^4}{3\Delta\left(0.61\Delta + z\right)^3}\right)\right] \tag{4.1}$$

Where z is the distance between a particle and the wall, Δ is the thickness of a single graphene layer, and σ_{sf} is a Lennard-Jones type parameter reflecting the size of the interaction between the fluid atom and a C atom in the graphene layers. The parameter A is a constant given by

$$A = 2\pi\rho_s\varepsilon_{sf}\sigma_{sf}^2\Delta \tag{4.2}$$

where ρ_s is the number density of C atoms in graphene and ε_{sf} is a Lennard-Jones type attraction parameter between the fluid particle and a C atom. A schematic diagram of the system is given in Fig 4.1. The parameter H in the diagram is the width of the slit pore. The total energy of an individual particle in the slit pore is then given by

$$V_{ext}(z) = \phi_{sf}(z) + \phi_{sf}(H - z) \tag{4.3}$$

4.2.2 Ewald calculation

Calculation of the Ewald energy for a two-dimensional system is different from the Ewald method presented in chapter 2 due to the lack of periodic images in the z direction. The two-dimensional Ewald summation is shown in Eqs. (4.4) – (4.7):

$$U_{realspace} = \frac{1}{2}\sum_{h_x,h_y}\sum_{i=1}^{N}\sum_{j=1}^{N}\frac{q_i q_j}{\left|r_{ij} + h_x + h_y\right|} \times erfc\left(\alpha\left|r_{ij} + h_x + h_y\right|\right) \tag{4.4}$$

$$U_{self} = \frac{-\alpha}{\sqrt{\pi}} \sum_{i=1} q_i^2 \tag{4.5}$$

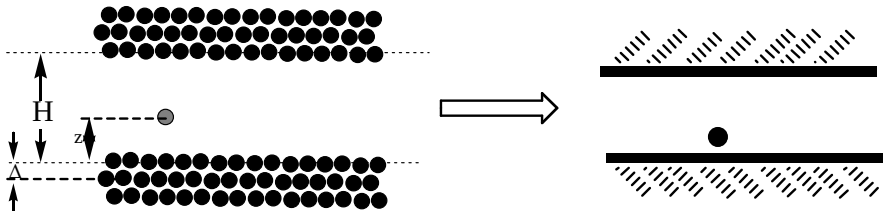


Fig. 4.1.1 Schematic diagram of the system representing a slit-shaped pore in graphene. The diagram on the left represents the fully atomic system that is represented in simplified form by the diagram on the right and by Eq. (4.1).

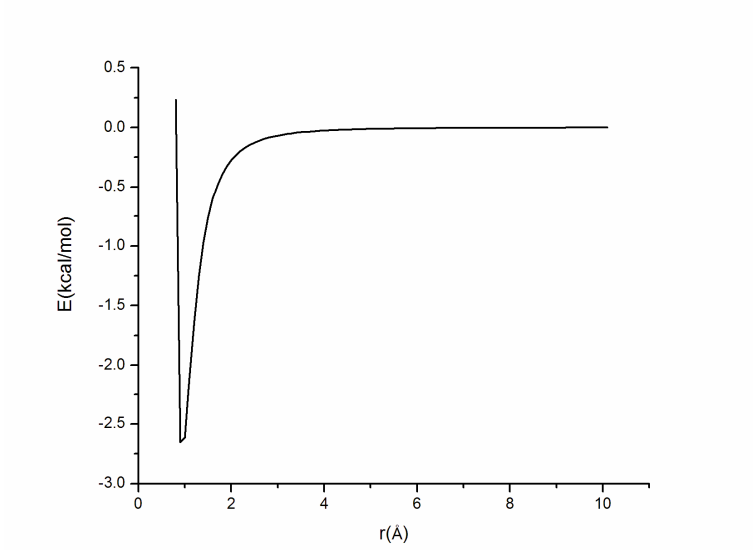


Fig. 4.1.2 Energy plot of particle in slit pore for methane by Eq. (4.1) and Eq. (4.2).

$$U_{reciprocal} = \frac{\pi}{2|h_x \times h_y|}\sum_{k\neq 0}\sum_{i=1}^{N}\sum_{i=1}^{N} q_i q_j \frac{\cos(k \cdot r_{ij}^{'})}{|k|} \times[e^{|k|z_{ij}} erfc(\frac{|k|}{2\alpha} + \alpha z_{ij}) + e^{-|k|z_{ij}} erfc(\frac{|k|}{2\alpha} - \alpha z_{ij})] \quad (4.6)$$

$$U_{intra} = -\frac{1}{|h_x \times h_y|}\sum_{i=1}^{N}\sum_{j=1}^{N} q_i q_j \times[\frac{\sqrt{\pi}}{\alpha} e^{-(\alpha z_{ij})^2} + \pi z_{ij} erf(\alpha z_{ij})] \quad (4.7)$$

Where r' indicates the projection of the position vector r onto the x-y plane. Unfortunately, implementation of this 2-dimensional form for the Ewald summation introduces substantial computational cost due to the loss of three-dimensional symmetry. An alternative method that is substantially more efficient is to treat the system as a 3-dimensional system with walls that are thick enough (>>3.35 Å) that periodic images of the slit pores in the *z*-direction are effectively non-interacting

4.3 Results and discussion

4.3.1 Program validation: Pure methane in a slit pore

Following Tan and Gubbins[31], we initially simulated pure methane in a slit pore. Conditions for these simulations are all expressed in terms of the following reduced parameters:

$$H^* = \frac{H}{\sigma_{ff}} \quad (4.7)$$

$$T^* = \frac{kT}{\varepsilon_{ff}} \tag{4.8}$$

$$\mu^* = \frac{\mu}{\varepsilon_{ff}} - 3T^* \ln\left(\frac{\Lambda}{\sigma_{ff}}\right) \tag{4.9}$$

$$\rho^* = \rho\sigma_{ff}^3 \tag{4.10}$$

$$z^* = \frac{z}{\sigma_{ff}} \tag{4.11}$$

Where σ_{ff} and ε_{ff} are methane Lennard-Jones parameters, μ is the methane chemical potential, z is the particle position in the z direction, ρ is the particle density in the slit pore, and $\Lambda = \sqrt{h^2/(2\pi m k_B T)}$ is the de Broglie wavelength. GCMC simulations were performed at a reduced temperature of $T^* = 1.35$ and a reduced chemical potential of $\mu^* = -5.00$ with reduced pore spacings of $H^* = 2.5$ and $H^* = 5.0$. The x and y dimensions for the system were taken to be 50 Å. The resulting density distribution for methane in the carbon slit pore, shown in Fig. 4.2, is essentially identical to the distributions reported by Tan and Gubbins[31]. Both systems show clear evidence of layered methane structures. Each density profile shows two large peaks corresponding to methane in contact with the wall, while the wider ($H^* = 5.0$) slit pore shows two smaller central peaks in addition.

A series of simulations was then performed for these same reduced pore spacings to reveal how methane density depends on the chemical potential. The measured average density of methane is plotted versus the ratio of reduced chemical potential to reduced temperature in Fig 4.3. The x-axis in the plot was

chosen to match that used by Tan and Gubbins even though the reduced temperature was held at a constant value of 1.35 in all simulations. The solid lines in the figure are the results from our program while the dashed lines are from Ref.31. Results for both pore spacings agree with those given by Tan and Gubbins[31]. The density necessarily increases with increasing chemical potential since the chemical potential is the thermodynamic variable that controls density changes. The density increases linearly with chemical potential for the narrow slit ($H^* = 2.5$) but shows a jump at around $\mu^*/T^* = 4$ for the $H^* = 5.0$ slit pore. Density profiles for the two slit pore systems are given as a function of reduced chemical potential in Fig. 4.4. From these figures, it is evident that the jump in density for the $H^* = 5.0$ slit pore system occurs in the region where filling of the central region of the slit pore is accomplished.

4.3.2 Pure water in a slit pore

GCMC simulations for pure water were performed at a reduced temperature of $T^* = 1.35$and reduced chemical potentials of $\mu^* = -39.95$ and -49.47 for reduced pore spacings of $H^* = 3.0$ and $H^* = 6.0$. Density profiles obtained from these simulations are displayed in Fig. 4.5. As for the methane system, there are two peaks evident for the narrow slit pore corresponding to water associated with the slit pore walls (Fig. 4.5.1). The density profiles show little dependence on chemical potential, suggesting that the system is essentially saturated by $\mu^* = -49.472$. For the wider slit pore H* = 6.0, the simulations still show two distinct peaks for water

adsorbed to the pore walls. In contrast to the methane results, however, the structure shows little evidence of layering in the central region (Fig. 4.5.2). This is likely due to the fact that a very low temperature was used for these initial water simulations rather than a temperature corresponding to liquid-phase water.

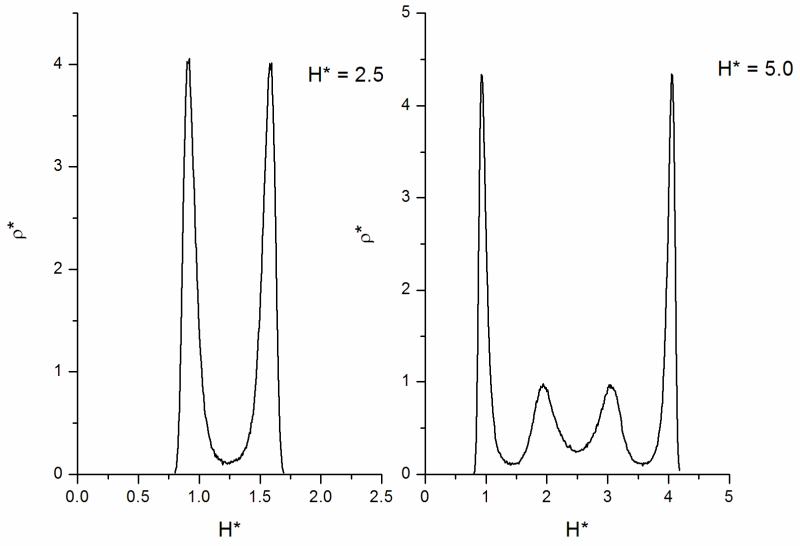


Fig. 4.2 Density profiles for methane in carbon slit pores at a reduced temperature of T^* = 1.35 and a reduced chemical potential of μ^* = -5.00.

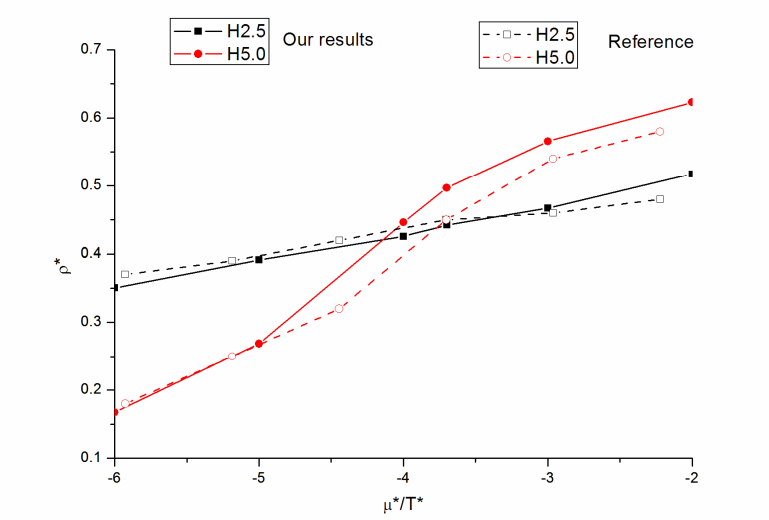


Fig. 4.3 Average reduced density of methane in carbon slit pores plotted as a function of the ratio of reduced chemical potential to reduced temperature. Results are shown for reduced pore spacings of H^* = 2.5 (squares) and H^* = 5.0 (circles). Filled symbols correspond to our work while open symbols are from Ref. 27.

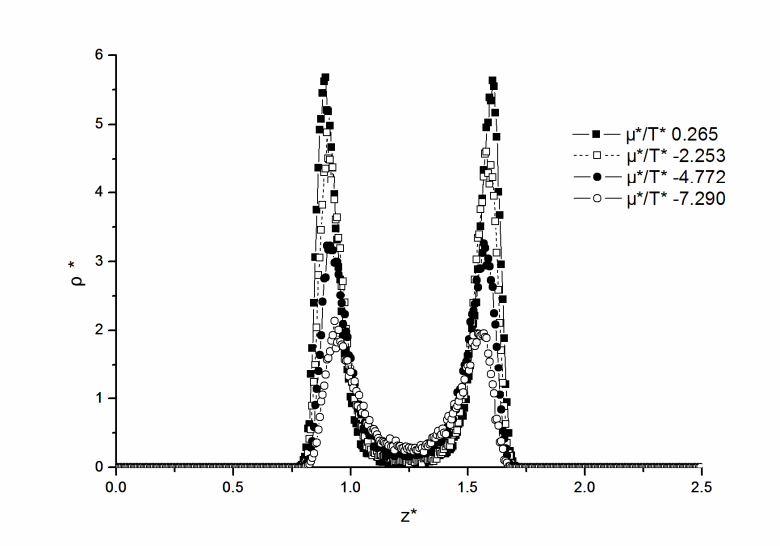


Fig. 4.4.1 Density profiles for methane in a carbon slit pores for several values of the reduced chemical potential, T^* = 1.35 and H^* = 2.5.

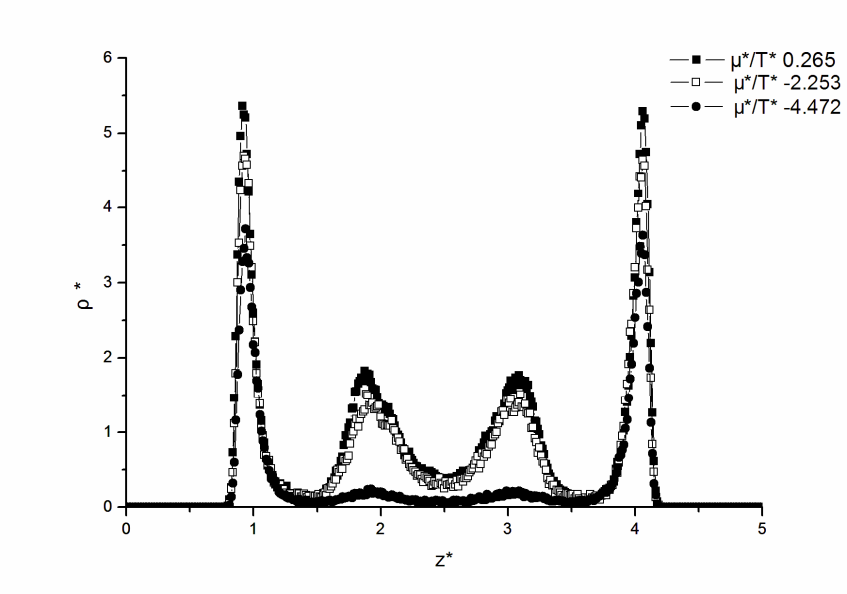


Fig. 4.4.2 Density profiles for methane in carbon slit pores for several values of the reduced chemical potential, T^* = 1.35 and H^* = 5.0

4.4 Future work

As noted in the chapter introduction, there is some experimental evidence that chathrate-hydrate formation plays a role in enhanced methane adsorption on pre-hydrated carbonaceous materials. A methane clathrate-hydrate is a stable, crystalline substance consisting of a water lattice within which methane molecules are trapped. Several different types of clathrate-hydrate lattices have been identified, including for example Type I, Type II, Type H and Type IX hydrates[33]. Methane clathrate hydrates are of a Type I structure which consists of five 5^{12} polyhedra (Fig. 4.6.1) and two $5^{12}6^2$ polyhedra (Fig. 4.6.2).[33]

Simulations of crystallization phenomena are notoriously difficult to perform[34]. It is therefore unlikely that a simulation would reveal spontaneous clathrate hydrate formation (on any reasonable time scale) even if simulation conditions could be tuned to favor their formation. Therefore, we propose the following simulation strategy.

1. Build a system containing a crystalline clathrate-hydrate structure by inserting known hydrate structures into a slit pore of appropriate width.
2. Perform Canonical ensemble (constant N, V, T) simulations to assess the stability of the clathrate hydrate system and to allow for equilibration away from the idealized starting configurations.
3. Investigate the temperature range of clathrate-hydrate stability by variation of the temperature.

4. Investigate how methane loading is impacted by the presence of the clathrate hydrate lattice using GCMC simulations in which the methane content is allowed to fluctuate while the water content is held constant. Comparison with GCMC simulations of the pure methane / slit pore system will then indicate whether any enhanced loading may be associated with the clathrate hydrate structures.

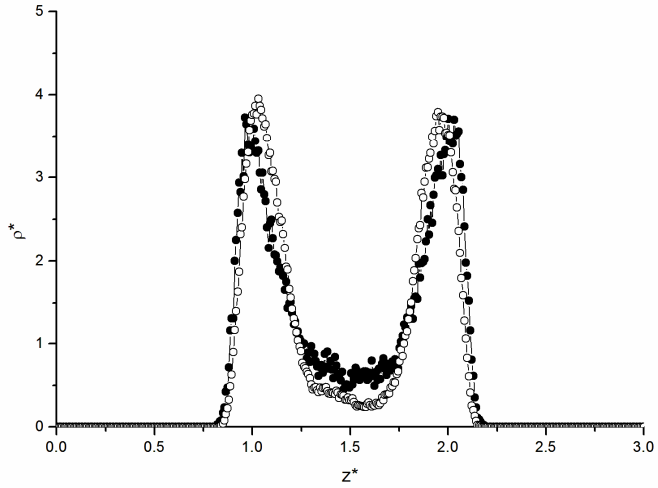


Fig. 4.5.1 Density profiles for water in a carbon slit pores at a reduced temperature of $T^* = 1.35$ and a reduced pore spacing of $H^* = 3.0$. The values of the reduced chemical potential are $\mu^* = -39.95$ (filled circles) and $\mu^* = -49.472$ (open circles).

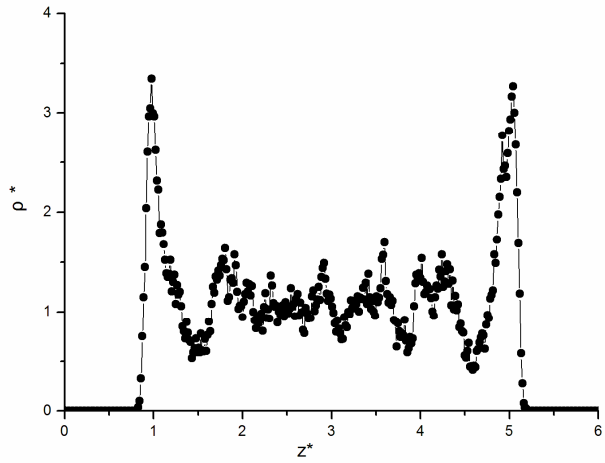


Fig. 4.5.2 Density profiles for water in a carbon slit pore at a reduced temperature of T^* = 1.35, a reduced pore spacing of H^* = 6.0, and a reduced chemical potential of μ^* = -39.95.

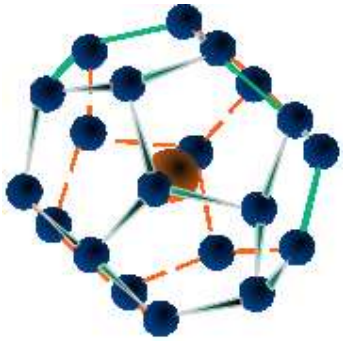


Fig. 4.6.1 Lattice 5^{12} in clathrate.

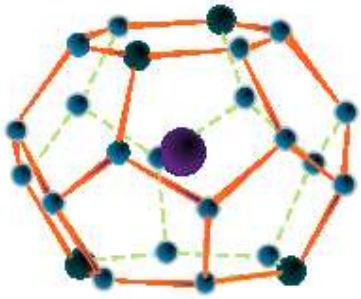


Fig. 4.6.2 Lattice $5^{12}6^{2}$ in clathrate.

4.5 References

[1] Springer, K.J. *Journal of Engineering for Gas Turbines and Power. 1992, 114*, 445.

[2] http://www.ldeo.columbia.edu/~orton/resources/OREGONIAN_energycrisis.pdf, visited at June 9, 2009.

[3] McShan, W.H.; Potter, V.R.; Goldman, A.; Shipley, E.G. *Am. J. Physiol 1945, 145*, 93.

[4] Gibbson, J.H. Energy from Biological Processes: Volume II – Technical and Environmental Analyses. U.S Government Printing Office, Washington, D.C., 1980.

[5] www.eere.energy.gov, visited at June 9, 2009.

[6] http://www.hydrogen.energy.gov/, visited at June 9, 2009.

[7] Ingersoll, J.G. *Natural gas vehicles.* Fairmont Press. Inc. 1996.

[8] Kowalczyk, P.; Tanaka, H.P. *Langmuir*. *2005, 21*, 5639.

[9] Zhou, L.; Sun, Y.; Zhou, Y.L. *AICHE Journal. 2002, 48*, 2412.

[10] Matusewicz, M.; Patrykiejew, A. *J. Chem. Phys. 2007, 127*, 174707.

[11] Quinn, D.F. *Carbon. 2002, 40*, 2767.

[12] Gusev, V.Y.; O'Brien, J.A.; Seaton, N.A. *Langmuir*. *1997, 13*, 3815.

[13] Peng, X.; Zhao, J.; Cao, D. *Journal of Colloid and Interface Science. 2007, 310*, 391.

[14] Murata, K.; Kaneko, K. *J. Phys. Chem. B. 2002, 106*, 11132.

[15] Anson, A.; Callejas, M. A. *Carbon. 2004, 42*, 1237.

[16] Kaneko, K.; Maedo, Y.; Okui, T. *Method of storing and transporting gases*. Japanese Patent JP *1996* 0037526 for Tokyo Gas.

[17] Tange, K. *Occluded natural gas storage method.* Japanese Patent JP *2000* 161595 for Toyota Motor Corp.

[18] Miyawaki, J.; Kanda, T.J. *J. Phys. Chem. B. 1998, 102,* 2187.

[19] Liu, X.; Zhou, L. *Carbon. 2006, 44*, 1386.

[20] Zhou, Y.; Wang, Y. *Carbon. 2005, 43*, 2007.

[21] Zhou, L.; Liu, X. *J. Phys. Chem. B. 2005, 109*, 22710.

[22] Zhou, Y.; Dai, M. *Carbon. 2004, 42*, 1855.

[23] Liu, X.; Zhou, L. *Journal of Porous Media. 2006, 9*, 769.

[24] Perrin, A.; Celzard, A. *Carbon. 2004, 42*, 1249.

[25] Perrin, A.; Celzard, A. *Energy & Fuels. 2003, 17*, 1283.

[26] Celzard, A.; Mareche, J.F. *Fuel. 2006, 85*, 957.

[27] Zhou, L. *Adsorption Science & Technology. 2005, 23*, 509.

[28] Muller, E.A.; Gubbins, K.E. *Carbon. 1998, 36*, 1433.

[29] Sizov, V.V; Piotrovskaya, E.M. Brodskaya, E.N. *Physical Chemistry of Surface Phenomena. 2007, 81*, 1285.

[30] Do, D.D.; Do, H.D. Adsorption Science & Technology. *2003, 21*, 389.

[31] Tan, Z.; Gubbins, K.E. *J. Phys. Chem. 1990, 94*, 6061.

[32] Steele, W.A.; *Surf. Sci. 1973, 36*, 317.

[33] Sloan, E.D. Clathrate Hydrates of Natural Gases. Marcel Dekker, New York, 1990.

[34] Ding, K.; Chandler, D. Smithline, S.J.; Haymet, A.D. *Physical Review Letters. 1987, 59*, 1698.

Chapter 5

Programming

Over the course of the research described in this thesis, a significant effort was spent on developing a flexible, object-oriented simulation code for potential use in a wide range of applications. The code is written in the C++ programming language, and is modeled after the FORTRAN code previously written in Dr. Smith's research group. It is a general Monte Carlo (MC) code that can also perform grand canonical Monte Carlo (GCMC) simulations of smaller molecules. It is written so the system can include a solution phase involving several types of molecules simultaneously, each of which may contain several different atom types. In this respect, it is a significant generalization of the existing Fortran code which is limited to the treatment of water and up to two atomic or ionic solutes in its solution phase. Both the FORTRAN and C++ codes have, in addition, the capability of representing complicated solid lattices such as clay minerals as well as simplified solid structures such as slit-pore walls described in Chapter 4.

5.1 Motivation

There are two primary motivations for writing a new C++ program for this research rather than simply modifying the existing FORTRAN source code. The first is that C++ is the object-oriented programming language which allows for easy generalization to new molecular systems. For example, the FORTRAN code was used initially for the chloroform/CNT simulations described in Chapter 3. This original code was well suited for simulation of the aqueous CNT system and was

also highly optimized for computational efficiency. Significant modifications were required, however, for inclusion of chloroform molecules into the simulation. Debugging required substantial effort and the readability of the overall code suffered substantially after these significant changes. The prospect of further generalization of the FORTRAN code to treatment of other interesting systems became more daunting as the code complexity increased.

Object-oriented programming in which each molecule is treated as a separate type of object, with independent code fragments, is naturally suited for the generalizations we envisioned. The C++ programming language was chosen for this development given not only its object-oriented structure but also for its widespread use in scientific programming and general computational efficiency.

A second reason for pursuing a new programming language has to do with memory allocation issues. In the original FORTRAN code, large arrays were used to enhance efficiency, but the number of required arrays increased substantially with the number of different atom or molecule types. This led to significant memory limitations for systems such as the chloroform-water mixtures described in Chapter 3. Memory allocation is done differently in the C++ language than in FORTRAN, and the program structure also lends itself to smaller memory requirements. This is a significant benefit especially for more complex systems envisioned for future studies.

5.2 Program files

A copy of the C++ source code and sample input and output files are posted on the following webpage: web.nmsu.edu/~lchen/Molecule/index.htm and are also included in the Appendix. The webpage contains two folders as diagrammed in Fig 5.1. The first folder, "Exe," contains a set of input and output files corresponding to a simulation of pure water. These files are easily modified to represent other systems. The input files include:

- input.in, containing coordinates and model parameters for all atoms in the starting configuration for the system. A sample illustrating the format of this table is given in Table 5.1. The entry for each atom includes its name, specifying the atom type, its *x*, *y*, and *z* coordinates, its atomic charge, and its Lennard-Jones and parameters.
- model.in, including templates for each type of molecule that can be included in the system. For example, the template for water consists of three entries, one for its O atom and two for its H atoms, in the format used in the input.in file. The coordinates are specified to assure the molecule has the proper geometry. Insertion trials in the GCMC program consist of choosing the desired template from the model.in file, with random translational and rotational moves used to generate the coordinates for the insertion trial molecule.

- parameter.in, including various system-specific input parameters that define the composition of the system and thermodynamic parameters. A complete list of parameters for this file is given in Table 5.2.
- system.in, including several general input parameters controlling the length of the simulation and the type and frequency of output to be performed. A complete list of parameters for this file is given in Table 5.3.

The second folder, "Src," includes the source code. These files include

- main.cpp, the main portion of the code that executes the simulation.
- Atom.h which controls and describes the properties of the atoms.
- Molecule.h which controls and describes the properties of the molecules built up from the atoms in Atom.h
- System.h which defines the geometry of the three dimensional box in which the molecules are contained.

The process for performing a simulation is very simple. The source code is first compiled in the Src folder and transferred to the Exe folder. This file is then executed following adjustment of relevant input files to define the desired system parameters. The program will generate several output files:

- output.out, including the final coordinates of all atoms and molecules in the system. It has the same format as the input.in file.

- number.out, giving the number of molecules as a function of the number of simulation cycles. This allows for easy plotting of the particle number fluctuations in GCMC simulations.
- gr.out, giving the structural data used for generating g(r) plots.
- output.pdb, including the final coordinates of all atoms and molecules in the system given in the pdb format used for graphical analysis in the VMD software package[3].

Table 5.1 Input file, model, file and output file format, water is take as the example at the bottom.

Molecule name	atom number	mass	fix			
first atom name	x	y	z	charge	σ	ε
second atom name	x	y	z	charge	σ	ε
third atom name	x	y	z	charge	σ	ε

water	3	18.016	0			
O	-0.212095244	7.973608	0.740273	-15.4476	3.1655	0.1554
H	-1.207210664	7.887144	0.78791	7.7238	0	0
H	0.082794332	7.909364	-0.2131	7.7238	0	0

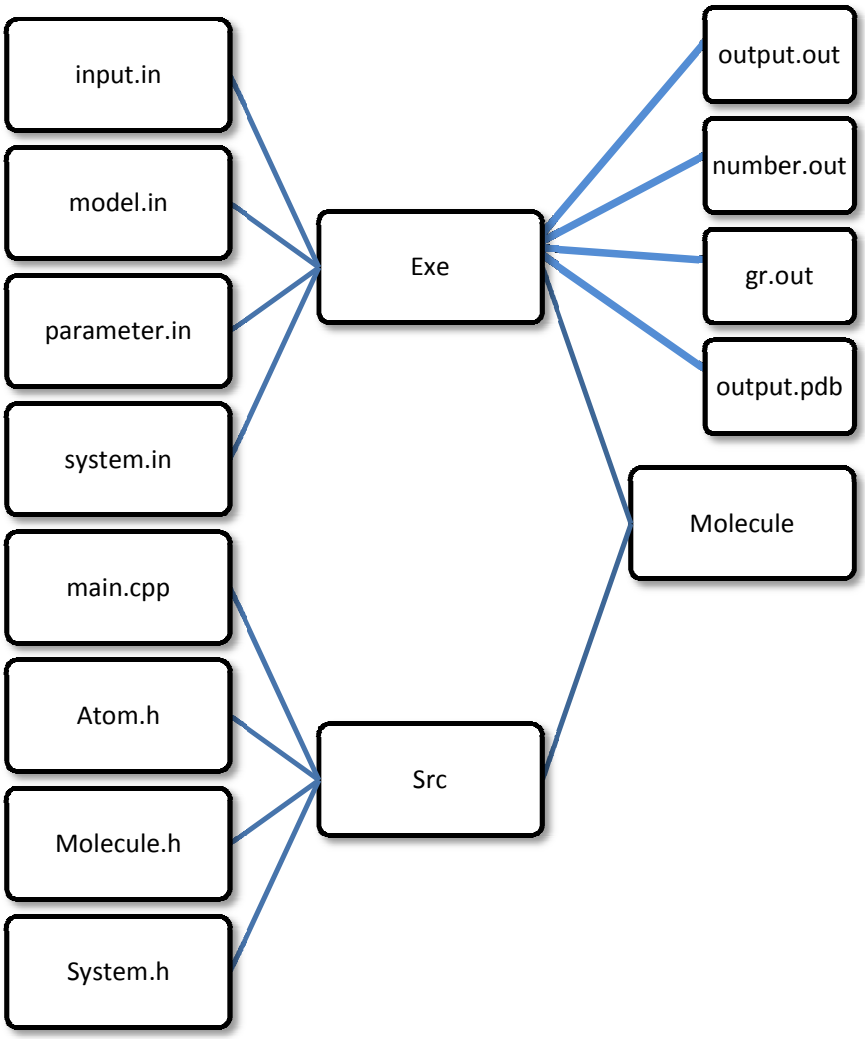


Fig. 5.1 Program folder structure.

Table 5.2 Parameters included in the parameter.in file in the Exe folder.

Parameter	Value	Comment
haveWall	0	if the box has wall
System_x	23.4905	size of the system at x direction
System_y	23.4905	size of the system at y direction
System_z	23.4905	size of the system at z direction
temperature	298.15	Temperature
moveStep	0.2	moving step in the movement trial
rotateStep	0.07	rotating step in the movement trial
lessParticle	0	reduce the molecule number in the system
notOnlyMove	1	1 insert, delete and movement trial, 0 movement trial
modGz	200	not define
modGr	100	not define
boundaryX	1	1 consider boundary at x direction, 0 not
thickX	3.35	thickness of wall at x direction
rhoX	0.114	constant parameter in 10-4-3 model at x directoin
sigWallX	3.4	sigma of the wall at x direction
epsWallX	0.0556	epsilon of the wall at x direction
boundaryY	1	1 consider boundary at y direction, 0 not
thickY	3.35	thickness of wall at y direction
rhoY	0.114	constant parameter in 10-4-3 model at y direction
sigWallY	3.4	sigma of the wall at y direction
epsWallY	0.0556	epsilon of the wall at y direction
boundaryZ	1	1 consider boundary at z direction, 0 not
thickZ	3.35	thickness of wall at z direction
rhoZ	0.114	constant paramter in 10-4-3 model at z direction
sigWallZ	3.4	sigma of the wall at z direction
epsWallZ	0.0556	epsilon of the wall at z direction
kMax	5	constant in ewald energy calculation
ksqMax	30	constant in ewald energy calculation
kappa	5.5	constant in ewald energy calculation
Slide	100	not define
Numpts	10000	not define
idratio	30	ratio of insertion to deletion
chemPotential	-11.61	potential of the system
checkTime	0	1 check the running time, 0 not
grPlot	1	1 plot the gr correlation function, 0 not
grResolution	100	resolution of gr plot

Table 5.3 Parameters included in the system.in file in the Exe folder.

Parameter	Value	Comment
cycle	1000000	total cycles that program run
insertNumber	350	inert molecule number if the system is empty
timeAnalysis	0	1 do the time analysis and generate the time table for every part, 0 not
moleculeNumberPlot	1	1 plot molecule number vs cycle, 0 not
moleculeNumberPlotStep	100	how many cycles to plot molecule nubmer vs cycle
checkFullEnergy	0	1 check the full energy of the system, 0 not
checkFullEnergyStep	1000	how many cycles to check the full energy one time
mediaOutput	1	1 output the coordinates during program running, 0 not
mediaOutputStep	1000	how many cycles to output the coordinates
grPlot	0	1 plot the gr correlation function, 0 not
grStep	1000	every how many cycles to plot the gr correlation function

5.3 Program structure

A flow chart for the GCMC program is given in Fig 5.2. Upon initialization, the program first checks to see if the system is empty or if some previous configuration is given as input. If the system is empty, a specified number of molecules will be inserted at random locations to obtain an initial configuration. Following initialization, the main portion of the program is executed. A random number is first generated to select whether an insertion, deletion, or movement trial will be attempted. The selected trial is then performed. Quantities that are collected over the course of the simulation, including the energy, particle number, structural information, etc., are updated and stored for later output. This process is then repeated. The simulation proceeds in this fashion until the number of cycles specified in the input parameters is achieved. The program then terminates with generation of the various output files.

The simulation system consists of a three dimensional box, as represented in Fig 5.3, with periodic boundary conditions. Conceptually, the system also includes a “molecule container” that holds all the molecules in the system. Molecules are built up from atoms that are included in an “atom container”. Movement trials and deletion trials randomly choose a molecule from the molecule container and operate on the molecule according to the type of trial chosen. Insertion trials generate a new molecule with random position and orientation by first selecting a template molecule from the “model container” and

then applying a random translation and rotation to this molecule before attempting the insertion. If the insertion is successful, the program will put the new molecule into the molecule container.

Input file, model file and output file have the format shown in Table 5.1.

5.4 Extensions from previous FORTRAN code

Several significant differences between the C++ and FORTRAN codes are highlighted in this section. The C++ source code is all included in the Appendix. First, the C++ program reorganizes the energy calculation functions. In the energy calculation, the Lennard-Jones potential and the real space part of Ewald summation are the only functions that depend on the distance between particles. They have been combined into a single function with the name “singleEnergy()” in System.h. The self energy term in the Ewald summation only depends on the molecular charge and is calculated by the function self() that is also included in System.h. The function intra() calculates the intramolecular potential energy and is contained within Molecule.h. It was previously included in the real-space energy calculation in the FORTRAN code but is separated here to improve computational efficiency.

The reciprocal space portion of the Ewald summation is the single most computationally expensive part of a simulation, typically constituting some 70% of the computational time. Significant optimization of this calculation resulting from symmetry considerations has been incorporated into both the FORTRAN and C++

source codes. Optimized single-molecule Ewald calculations are also incorporated for use in the insertion, deletion and movement trials where the change in energy associated with a single molecule transformation is required. The parameter r_x, r_y and r_z that are the r value at x, y, z direction is added to each molecule. The reciprocal part of the Ewald summation can be rewritten as:

$$U_{reciprocal} = \frac{1}{\varepsilon_o V}\sum_{k>0}\frac{1}{k^1}e^{-\frac{k^2}{4\alpha^2}} \times\left(|\sum_{i=1}^{N} q_i \cos(k\cdot r_i)|^2 + |\sum_{i=1}^{N} q_i \sin(k\cdot r_i)|^2\right) \tag{5.1}$$

where the parameters have same meaning as in equation (2.20)[1]. The terms immediately following the summation involving k and α are constant for constant V simulations and are calculated only once in the ksetup() function in System.in. This differs form the FORTRAN code in which these terms are recalculated regularly in order to support system volume changes.

The Lennard-Jones and real-space portion of the Coulomb energy are calculated in the FORTRAN code using large potential energy arrays in order to avoid many time-consuming energy calculations over the course of a simulation. This strategy was abandoned for the C++ code for two reasons. Firstly, using a real-time calculation of the energy functions dramatically reduces the memory requirements, especially for complex systems containing many different atom types. Memory bottlenecks are thus avoided. Secondly, a real-time energy calculation eliminates the interpolation error associated with the use of energy tables

(i.e., arrays) in the energy determination. This error was especially notable in the intramolecular energy determination due to the short interatomic distances in this calculation. For example, the intramolecular energy for a single water molecule calculated by the FORTRAN and C++ codes are -119.3232 kcal/mol and -119.3229 kcal/mol, respectively. The improvement is small but potentially significant. The benefits of real-time energy calculations, particularly with respect to memory use, are significant enough to justify the slight reduction in computationally efficiency that is also associated with this approach.

Finally, several new features have been incorporated into the C++ code. First, a new parameter was defined to specify whether a molecule should be fixed in place or not during the course of a simulation. This is used both for rigid entities such as the CNT lattice and also for molecules constrained for the purposes of thermodynamic integration calculations such as those described in Chapter 3. In contrast, fixing individual molecules in place required substantial code modifications in the FORTRAN code. In addition, the FORTRAN program was generalized for treatment of featureless walls using, for example, Steele's 10-4-3 model[2] as described in Chapter 4. Finally, a new feature allowing generation of output files in pdb (protein data base) format allows or easy visualization of simulation configurations using VMD[3] or other visualization software.

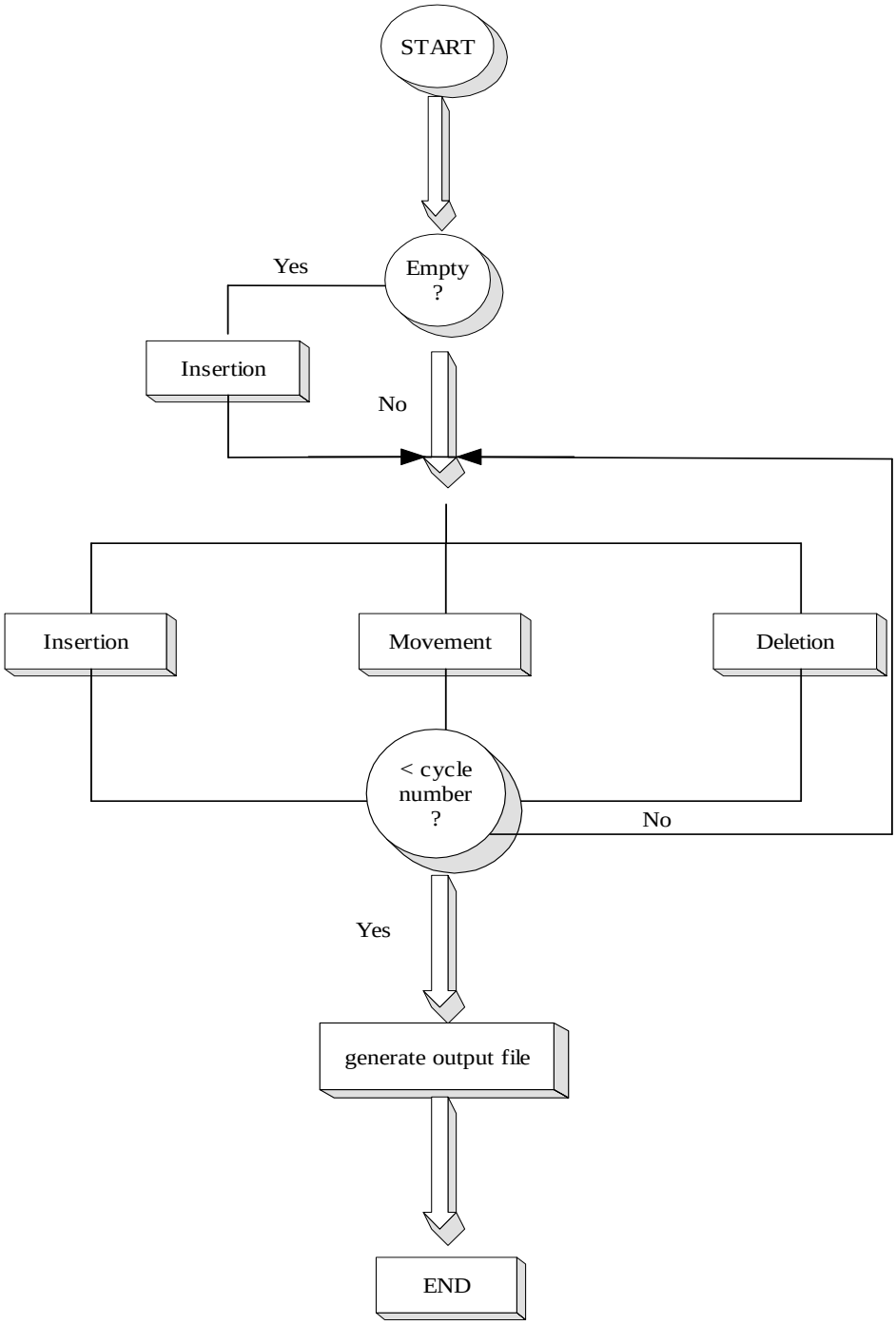


Fig. 5.2 Program flow chart.

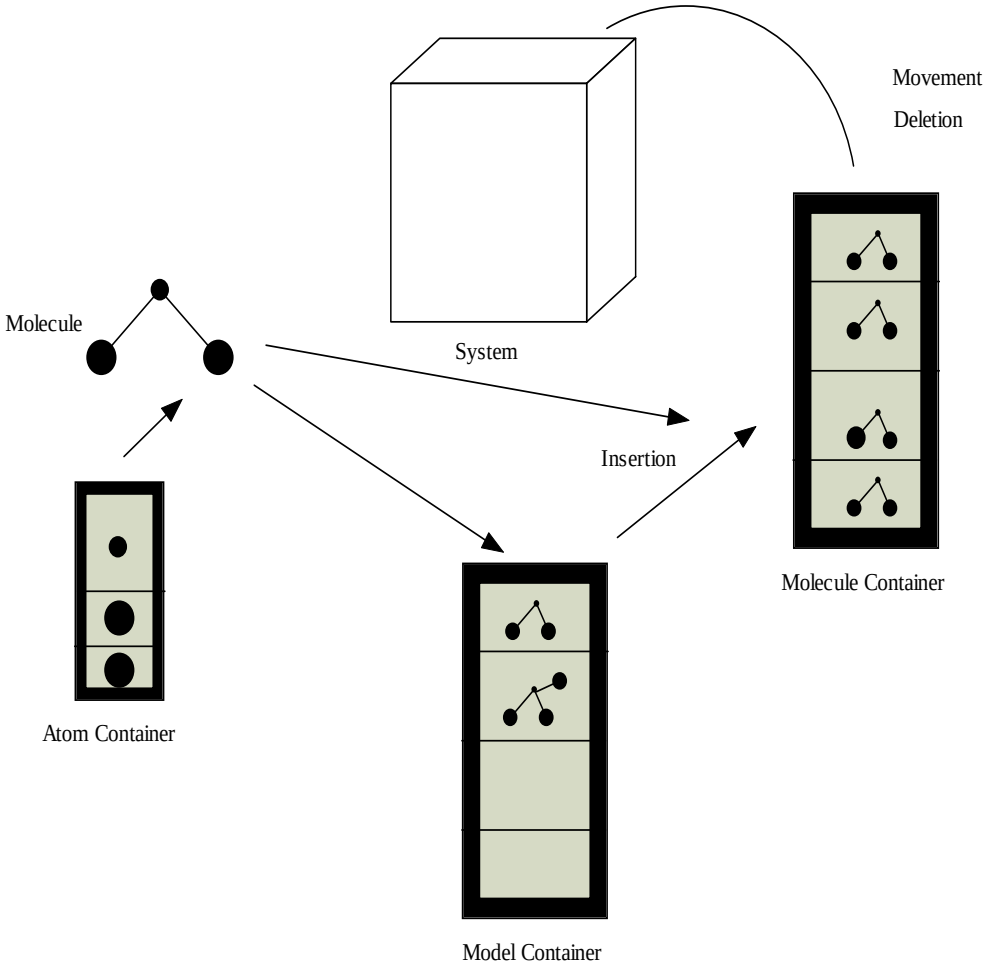


Fig. 5.3 System structure.

5.5 References

[1] *Understanding Molecular Simulation from Algorithms to Applications*; chapter 12.

[2] Steele, W.A.; *Surf. Sci. 1973, 36,* 317.

[3] www.ks.uiuc.edu/Research/vmd/, visited on July 15, 2009.

APPENDICES

APPENDIX A

MONTE CARLO SIMULATION PROGRAM FOR SMALL MOLECULE

```
/*Program "MOLECULE"
 * Lin Chen
 * Jan. 1, 2009
 *
 * This program is a general program that can execute the small molecule system *
 simulaiton by Monte Carlo method It is derived from Dr. Smith's Fortran code *
 for water
 * main.cpp input the set up from system.in
 * System.h design a three dimension box, it can represent the whole space by
    boundary condition
 * varies of molecules can be inserted into this box and execute the simulation by
    insertion, deletion and movement
 * Molecule.h design the molecule that can be inserted into the system
 * Atom.h design the atom that can build up the molecule
 */

/*main.cpp*/
#include <iostream>
#include <fstream>
#include <vector>
#include <ctime>
#include "Molecule.h"
#include "System.h"
using namespace std;

int main()
{
    //Create a system
    System s;

    //read the system set up
    ifstream inputFile("system.in");
    int cycle;
    string str;
    //read the cycle that system runs
    inputFile>>str>>cycle;
   if(str != "cycle")
        cout<<"Parameter idratio error!"<<endl;

    int insertNumber;
    //read the particle number that can be inserted when the system is empty
```

```
    inputFile>>str>>insertNumber;
    if(str != "insertNumber")
        cout<<"Parameter insertNumber error!"<<endl;

    cout<<"Simulation will run "<<cycle<<" cycles, be patient..."<<endl;
    cout<<endl;
    cout<<"----------------------------------------"<<endl;

    int timeAnalysis;
    //read the timeAnalysis, 1 do the analysis, 0 do not do the analysis
    inputFile>>str>>timeAnalysis;
    if(str != "timeAnalysis")
        cout<<"Parameter timeAnalysis error!"<<endl;

    int moleculeNumberPlot, moleculeNumberPlotStep;
    //read the moleculeNumberPlot, 1 plot the molecule number vs cycle, 0 do not
plot
    inputFile>>str>>moleculeNumberPlot;
    if(str != "moleculeNumberPlot")
        cout<<"Parameter moleculeNumberPlot error!"<<endl;
    //read the plot step
    inputFile>>str>>moleculeNumberPlotStep;
    if(str != "moleculeNumberPlotStep")
        cout<<"Parameter moleculeNumberPlotStep error!"<<endl;

    int checkFullEnergy, checkFullEnergyStep;
    //read the checkFullEnergy, 1 check the system energy, 0 do not check
    inputFile>>str>>checkFullEnergy;
    if(str != "checkFullEnergy")
        cout<<"Parameter checkFullEnergy error!"<<endl;
    //read the energy check step
    inputFile>>str>>checkFullEnergyStep;
    if(str != "checkFullEnergyStep")
        cout<<"Parameter checkFullEnergyStep error!"<<endl;
   int mediaOutput, mediaOutputStep;
    //read the mediaOutput, 1 output the coordinates of molecules during running,
0 only output the coordinates when the running is done
    inputFile>>str>>mediaOutput;
    if(str != "mediaOutput")
        cout<<"Parameter mediaOutput error!"<<endl;
    //read the output step
```

```
inputFile>>str>>mediaOutputStep;
if(str != "mediaOutputStep")
    cout<<"Parameter mediaOutputStep error!"<<endl;

int grPlot, grStep;
//read the grPlot, 1 plot the gr function, 0 do not plot
inputFile>>str>>grPlot;
if(str != "grPlot")
    cout<<"Parameter grPlot error!"<<endl;
//read the gr plot step
inputFile>>str>>grStep;
if(str != "grStep")
    cout<<"Parameter grStep error!"<<endl;

double averageEnergy;//represent the average energy of molecule in the system
int averageEnergyCount;//accumulate the energy
//initialize the averageEnergy and averageEnergyCount
if(checkFullEnergy == 1)
{
    averageEnergy = 0.0;
    averageEnergyCount = 0;
}

ofstream numberPlot;
//open number.out to output the number vs cycle
if(moleculeNumberPlot == 1)
    numberPlot.open("number.out");

time_t start;
//record the start time
if(timeAnalysis == 1)
    time(&start);

int idratio;
idratio = s.getIdratio();

int nfi = 0;
double randomNumber;

//start the cycle
while(nfi <= cycle*(idratio+2))
{
```

```
        //system can do insertion, deletion and movement
        if(s.getNotOnlyMove())
        {
        //insert some molecules to get a initial configuration when the system is
emtpy
        if(s.getMoleculeNumber() == 0)
        {
            cout<<"Initial molecule number is zero, insert "<<insertNumber<<"
molecules firstly to\n obtain a start configuration..."<<endl;
            while(s.getMoleculeNumber() < insertNumber)
                s.insertion();
        }
        randomNumber = s.ran();
        if(randomNumber < double(idratio)/double(idratio+2))
        {
            //do the insert trial
            s.insertion();
        }
        else if(randomNumber < double(idratio+1)/double(idratio+2))
        {
            //do the delete trial
            s.deletion();
        }
        else
        {
            //do the movement trial
            s.movement();
        }
        nfi++;
        }
        //only do the movement trial
        else
        {
            //do the movement trial
            s.movement();
            nfi += 3;
        }

        //output the number of molecule vs cycle
        if(moleculeNumberPlot == 1)
        {
```

```
            int tempcycle;
            tempcycle = nfi/(idratio+2);
            if(nfi%(moleculeNumberPlotStep*(idratio+2)) == 0)
                s.moleculeNumberPlot(tempcycle, numberPlot);
        }

        //calculate the full energy of system
        if(checkFullEnergy == 1)
        {
            if(nfi%(checkFullEnergyStep*(idratio+2)) == 0)
            {
                averageEnergy += s.averageFullEnergy();
                averageEnergyCount++;
            }
        }

        //output the coordiante
        if(mediaOutput == 1)
        {
            if(nfi%(mediaOutputStep*(idratio+2)) == 0)
            {
                ofstream outputFile;
                outputFile.open("output.out");
                s.save(outputFile);
                outputFile.close();
            }
        }
    //plot the gr function
        if(grPlot == 1)
        {
            if(nfi%(grStep*(idratio+2)) == 0)
            {
                ofstream grOutput;
                grOutput.open("gr.out");
                s.gr(grOutput);
                grOutput.close();
            }
        }
    }

    //close number.out file
```

```
    if(moleculeNumberPlot == 1)
        numberPlot.close();

    time_t end;
    //check the end time
    if(timeAnalysis == 1)
    {
        s.timeAnalysis();
        time(&end);
        cout<<"Program run "<<end - start<<" seconds or "<<double(end - start)/3600<<" hours."<<endl;
    }

    //generate pdb file for molecules in the system
    s.generatePdb();

    cout<<"Average energy of system is: "<<averageEnergy/averageEnergyCount<<endl;

    cout<<"Successfully run."<<endl;

    return 0;
}
/*System.h*/
/*descript a system that is a three-dimension box which can contain varies of molecules*/
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib>
#include <ctime>
#include <vector>
#include <string>
#include "Molecule.h"
using namespace std;

class System
{
    private: double system_x, system_y, system_z;//system size at x, y and z direction
             double wallmin;//minimum value of system_x, system_y and system_z
```

```
        double hplanck;//planck constant
        double kboltzmann;//boltzmann constant
        double temperature;//represent the temperature in the system
        double twoPi;//2*pi
        double piSqr;//pi^2
        double rSqrtPi;//1/pi^2
        int mvec;
        double moveStep, rotateStep;//move step and rotate step in the movement
trial
        int lessParticle;//reduce the system particle
        bool haveWall, notOnlyMove;//haveWall, 1 has the wall, 0 do no have
wall; notOnlyMove, 1 can insert, delete and movement trial, 0 only can do the
movement trial
        int modGz, modGr;
        double thickOfWall, factorWall;//thickOfWall, parameter of the wall;
factorWall, factor of steel's 10-4-3 model
        bool boundaryX, boundaryY, boundaryZ;//1 consider the boundary
condition, 0 do not consider the boundary condition
        double thickX, thickY, thickZ;//thick of the walls at x, y and z direction
        double sigWallX, sigWallY, sigWallZ;//sigma of the walls
        double epsWallX, epsWallY, epsWallZ;//epsilon of the walls
        double rhoX, rhoY, rhoZ;//parameters of the wall in the 10-4-3 model
        double ewaldTempX, ewaldTempY, ewaldTempZ;

       int kMax, ksqMax, slide;
        double kappa;//constant in the ewald potential
        int numpts;
        int idratio;//ratio of insert to delete
        double chemPotential;//represent the potential of the system
        vector<Molecule> modelContainer, moleculeContainer;//model
container and molecule container
        double KbT;//kbT
        vector<int> ihx, ihy, ihz;//container for ihx, ihy, and ihz that are mediate
value during ewald calculation
        vector<double> fhold;//constant part in the reciprocal space term in the
ewald potential
        vector<double> ssum, csum, ssum2, csum2;//sum of sin and cos in the
ewal potential
        int checkTime;//if check the running time, 1 check, 0 do not check
        time_t insertReal, insertReciprocal, insertTotal;//chekc the insert part
time
```

```
        time_t moveReal, moveReciprocal, moveTotal;//check the movement
part time
        time_t deleteReal, deleteReciprocal, deleteTotal;//check the delete part
time
        bool grPlot;//plot the gr function
        int grColumn, grRow, grResolution;
        double **grArray;
        int grCount, *grCountNum;
    public:
        //Constructor
        System()
        {
            //set the constant
            hplanck = 6.62608e-34;
            kboltzmann = 1.38066e-23;
            twoPi = 6.2831853;
            piSqr = 9.869604401;
            rSqrtPi = 0.564189583;
            mvec = 1000;
            string str;
            ifstream inputMolecule;
            //open input.in to read in the coordinates of the molecules in the
system
            inputMolecule.open("input.in");
            ifstream inputParameter;
            //open parameter.in to read in the parameters in the system
            inputParameter.open("parameter.in");
            ifstream inputModel;
            //open model.in to read in the information of the molecules that can
be inserted into the system
            inputModel.open("model.in");
            unsigned seed;
            //give random seed
            seed = time(0);
            srand(seed);
            //read the information of the molecules in the system
            //read the information of the model molecules
            //read the system parameters
            read(inputMolecule, inputModel, inputParameter);
            //get the wallmin
            if(haveWall)
            {
```

```
        if(!boundaryX)
        {
            wallmin = system_x + thickX;
        }
        else
        {
            wallmin = system_x;
        }
        if(!boundaryY)
        {
            if(system_y + thickY < wallmin)
                wallmin = system_y + thickY;
        }
        else
        {
            if(system_y < wallmin)
                wallmin = system_y;
        }
        if(!boundaryZ)
        {
            if(system_z + thickZ < wallmin)
                wallmin = system_z + thickZ;

    }
        else
        {
            if(system_z < wallmin)
                wallmin = system_z;
        }
    }
    else
    {
        wallmin = system_x;
        if(system_y < wallmin)
            wallmin = system_y;
        if(system_z < wallmin)
            wallmin = system_z;
    }
    KbT = 0.00831451/4.184*temperature;
    //initialize the k value in the ewald potential
    ksetup();
```

```
    //calculate the initial ewal potential
    ewald();
    //close the input files
    inputMolecule.close();
    inputModel.close();
    inputParameter.close();
    //initialize the time if the checkTime is 1
    if(checkTime == 1)
    {
        insertReal = 0;
        insertReciprocal = 0;
        insertTotal = 0;
        moveReal = 0;
        moveReciprocal = 0;
        moveTotal = 0;
        deleteReal = 0;
        deleteReciprocal = 0;
        deleteTotal = 0;
    }
    //initialize the value if the grPlot is 1
    if(grPlot)
    {
        int tempNum;
        double systemRange;
        tempNum = modelContainer.size();
        grColumn = (1+tempNum)*tempNum/2;
        systemRange = sqrt(system_x*system_x+system_y*system_y);
        grRow = int(systemRange*grResolution);
        grArray = new double *[grRow];
        for(int i = 0; i < grRow; i++)
            grArray[i] = new double [grColumn];
        for(int i = 0; i < grRow; i++)
            for(int j = 0; j < grColumn; j++)
                grArray[i][j] = 0.0;
        grCount = 0;
        grCountNum = new int [modelContainer.size()];
        for(int i = 0; i < modelContainer.size(); i++)
            grCountNum[i] = 0;
    }

}
```

```
        //Deconstructor
        ~System()
        {
            ofstream outputMolecule;
            //open output.out to output the coordinates of the molecule in the
system
            outputMolecule.open("output.out");
            save(outputMolecule);
            //close the ouput file
            outputMolecule.close();
            //delete the pointer
            if(grPlot)
            {
                for(int i = 0; i < grRow; i++)
                    delete [] grArray[i];
                delete [] grArray;
            }
        }

        //Mutator
        //read the molecule, model molecule and parameters
        void read(ifstream &inputMolecule, ifstream &inputModel, ifstream
&inputParameter)
        {
            string str;
            inputParameter>>str>>haveWall;
            if(str != "haveWall") cout<<"Parameter haveWall error!"<<endl;
            inputParameter>>str>>system_x;
            if(str != "system_x") cout<<"Parameter system_x error!"<<endl;
            inputParameter>>str>>system_y;
            if(str != "system_y") cout<<"Parameter system_y error!"<<endl;
            inputParameter>>str>>system_z;
            if(str != "system_z") cout<<"Parameter system_z error!"<<endl;
            inputParameter>>str>>temperature;
            if(str != "temperature") cout<<"Parameter temperature
error!"<<endl;
            inputParameter>>str>>moveStep;
            if(str != "moveStep") cout<<"Parameter moveStep error!"<<endl;
            inputParameter>>str>>rotateStep;
            if(str != "rotateStep") cout<<"Parameter rotateStep error!"<<endl;
            inputParameter>>str>>lessParticle;
```

```
		if(str != "lessParticle") cout<<"Parameter lessParticle error!"<<endl;
		inputParameter>>str>>notOnlyMove;
		if(str != "notOnlyMove") cout<<"Parameter notOnlyMove
error!"<<endl;
		inputParameter>>str>>modGz;
		if(str != "modGz") cout<<"Parameter modGz error!"<<endl;
		inputParameter>>str>>modGr;
		if(str != "modGr") cout<<"Parameter modGr error!"<<endl;
		inputParameter>>str>>boundaryX;
		if(str != "boundaryX") cout<<"Parameter boundaryX error!"<<endl;
		inputParameter>>str>>thickX;
		if(str != "thickX") cout<<"Parameter thickX error!"<<endl;
		inputParameter>>str>>rhoX;
		if(str != "rhoX") cout<<"Parameter rhoX error!"<<endl;
		inputParameter>>str>>sigWallX;
		if(str != "sigWallX") cout<<"Parameter sigWallX error!"<<endl;
		inputParameter>>str>>epsWallX;
		if(str != "epsWallX") cout<<"Parameter epsWallX error!"<<endl;
		inputParameter>>str>>boundaryY;
		if(str != "boundaryY") cout<<"Parameter boundaryY error!"<<endl;
		inputParameter>>str>>thickY;
		if(str != "thickY") cout<<"Parameter thickY error!"<<endl;
		inputParameter>>str>>rhoY;
		if(str != "rhoY") cout<<"Parameter rhoY error!"<<endl;
		inputParameter>>str>>sigWallY;
		if(str != "sigWallY") cout<<"Parameter sigWallY error!"<<endl;
		inputParameter>>str>>epsWallY;
		if(str != "epsWallY") cout<<"Parameter epsWallY error!"<<endl;
		inputParameter>>str>>boundaryZ;
		if(str != "boundaryZ") cout<<"Parameter boundaryZ error!"<<endl;
		inputParameter>>str>>thickZ;
		if(str != "thickZ") cout<<"Parameter thickZ error!"<<endl;
		inputParameter>>str>>rhoZ;
		if(str != "rhoZ") cout<<"Parameter rhoZ error!"<<endl;
		inputParameter>>str>>sigWallZ;
		if(str != "sigWallZ") cout<<"Parameter sigWallZ error!"<<endl;
		inputParameter>>str>>epsWallZ;
		if(str != "epsWallZ") cout<<"Parameter epsWallZ error!"<<endl;
		inputParameter>>str>>kMax;
		if(str != "kMax") cout<<"Parameter kMax error!"<<endl;
		inputParameter>>str>>ksqMax;
```

```
		if(str != "ksqMax") cout<<"Parameter ksqMax error!"<<endl;
		inputParameter>>str>>kappa;
		if(str != "kappa") cout<<"Parameter kappa error!"<<endl;
		inputParameter>>str>>slide;
		if(str != "slide") cout<<"Parameter slide error!"<<endl;
		inputParameter>>str>>numpts;
		if(str != "numpts") cout<<"Parameter numpts error!"<<endl;
		inputParameter>>str>>idratio;
		if(str != "idratio") cout<<"Parameter idratio error!"<<endl;
		inputParameter>>str>>chemPotential;
		if(str != "chemPotential") cout<<"Parameter chemPotential
error!"<<endl;
		inputParameter>>str>>checkTime;
		if(str != "checkTime") cout<<"Parameter checkTime error!"<<endl;
		inputParameter>>str>>grPlot;
		if(str != "grPlot") cout<<"Parameter grPlot error!"<<endl;
		inputParameter>>str>>grResolution;
		if(str != "grResolution") cout<<"Parameter grResolution
error!"<<endl;
		Molecule temp;
		int end;
		//check the size of the model.in file
		inputModel.seekg(0L, ios::end);
		end = inputModel.tellg();
		inputModel.seekg(0L, ios::beg);
		wallThick();
		//read out the data if the file is not empty
		if(end > 0)
		{
			while(inputModel.tellg() < end - 1)
			{
				temp.clear();
				temp.read(inputModel, twoPi, ewaldTempX, ewaldTempY,
ewaldTempZ);
				modelContainer.push_back(temp);
			}
		}
		//check the size of the input.in file
		inputMolecule.seekg(0L, ios::end);
		end = inputMolecule.tellg();
		inputMolecule.seekg(0L, ios::beg);
```

```
            //read out the molecule information if the file is not empty
            if(end > 0)
            {
                while(inputMolecule.tellg() < end - 1)
                {
                    temp.clear();
                    temp.read(inputMolecule, twoPi, ewaldTempX,
ewaldTempY, ewaldTempZ);
                    moleculeContainer.push_back(temp);
                }
            }
            //output the program start information
            cout<<"Program start..."<<endl;
            cout<<"------------------------------------------------"<<endl;
            cout<<"Read "<<modelContainer.size()<<" molecule models from
\"model.in\"."<<endl;
            for(int i = 0; i < modelContainer.size(); i++)
                cout<<"      "<<"No."<<setw(3)<<i+1<<"
"<<modelContainer[i].getName()<<endl;
            cout<<"Read "<<moleculeContainer.size()<<" molecules from
\"input.in\"."<<endl;

cout<<"------------------------------------------------"<<endl;
            if(notOnlyMove == 1)
            {
            cout<<"Then will randomly choose model molecule to insert into
system."<<endl;
            cout<<"System can do movement and deletion operation by
randomly choosing\n molecule from molecules in the system."<<endl;
            }
            else
            {
                cout<<"Then will randomly choose molecule to implement
movement."<<endl;
            }
        }

        //save the molecules in the system to output.out
        void save(ofstream &outputMolecule)
        {
            for(int i = 0; i < moleculeContainer.size(); i++)
            {
```

```
                moleculeContainer[i].save(outputMolecule);
            }
        }

        //movement trial, the step has be checked in the previous code, current parameters can obtain good MSD value for water
        void movement()
        {
            //return if the system is empty
            if(moleculeContainer.size() == 0)
            {
                cout<<"Molecle number in the system is zero, can not do movement operation."<<endl;
                return;
            }
            //check the movement trial time
            time_t start, end, startTotal, endTotal;
            if(checkTime == 1)
                time(&startTotal);
            int choice, last;
            //randomly choose a molecule to move
            choice = rand()%moleculeContainer.size();

            //return if the molecule choosed can not be moved
            if(moleculeContainer[choice].getFixing()) return;

            //choose the last molecule in the system
            last = moleculeContainer.size() - 1;

            exchange(choice);//exhange the choosed particle and last particle in the container

            Molecule temp;
            //save the choosed molecule
            temp = moleculeContainer[last];

            double energyBeforeMove;
            if(checkTime == 1)
                time(&start);
            //check the Lennard-Jones potential and real space energy before movement trial
```

```
                energyBeforeMove = singleEnergy();
                if(checkTime == 1)
                {
                    time(&end);
                    moveReal += end - start;
                }

                double reciprocalEnergy = 0;
                bool isBefore;
                isBefore = true;
                if(checkTime == 1)
                    time(&start);
                //check the reciprocal energy before movement trial
                ewaldMove(isBefore, reciprocalEnergy);
                if(checkTime == 1)
                {
                    time(&end);
                    moveReciprocal += end - start;
                }

                //shift the choosed molecule
                moleculeContainer[last].shift(moveStep, twoPi, ewaldTempX,
ewaldTempY, ewaldTempZ);
                //rotate the choosed molecule
                moleculeContainer[last].tinyRotate(twoPi, rotateStep, ewaldTempX,
ewaldTempY, ewaldTempZ);

                double energyAfterMove;
                if(checkTime == 1)
                    time(&start);
                //check the Lennard-Jones potential and real space energy after
movement trial
                energyAfterMove = singleEnergy();
                if(checkTime == 1)
                {
                    time(&end);
                    moveReal += end - start;
                }
                isBefore = false;
                if(checkTime == 1)
                    time(&start);
```

```
            //check the reciprocal energy after movement trial
            ewaldMove(isBefore, reciprocalEnergy);
            if(checkTime == 1)
            {
                time(&end);
                moveReciprocal += end - start;
            }

            energyAfterMove += reciprocalEnergy;

            double differenceEnergy;
            //check the energy difference after movement trial
            differenceEnergy = energyAfterMove - energyBeforeMove;


            //accetp the movement trial if the energy is reduced or random
number less than the accept factor
            if(differenceEnergy < 0 || ran() < exp((-1/KbT)*differenceEnergy))
            {
                //cancel the movement if the movement makes the molecule out
of the system from the wall direction
                if(moleculeContainer[last].outsideTell(system_x, system_y,
system_z, haveWall, boundaryX, boundaryY, boundaryZ))
                {
                    ewaldMoveUnexe(choice);
                }
                //move the molecule back into the system if the movement
makes the molecule is out of system
                moleculeContainer[last].align(system_x, system_y, system_z,
twoPi, ewaldTempX, ewaldTempY, ewaldTempZ);
                //renew the ewald and exchange the choosed particle back to the
initial position in the sequence
                ewaldMoveExe(choice);
            }
            //change the choosed molecule to intial coordinates
            else
            {
                moleculeContainer[last] = temp;
                //restore the ewald and exchange the choosed particle back to the
initial position in the sequence
                ewaldMoveUnexe(choice);
            }
```

```
            //check the time comsumption in the movement trial
            if(checkTime == 1)
            {
                time(&endTotal);
                moveTotal += endTotal - startTotal;
            }

        }

        //deletion trial
        void deletion()
        {
            //return if the system is empty
            if(moleculeContainer.size() == 0)
            {
                cout<<"Molecule number in the system is zero, can not do
deletion operation."<<endl;
                return;
            }
            time_t start, end, startTotal, endTotal;
            //check the delete trial time
            if(checkTime == 1)
                time(&startTotal);
            int choice, last;
            //random choose the molecule
            choice = rand()%moleculeContainer.size();

            //return if the molecule can not be deleted
            if(moleculeContainer[choice].getFixing()) return;

            last = moleculeContainer.size() - 1;

            double mass;
            mass = moleculeContainer[choice].getMass();

            double thermalWaveLength;
            //calculate thermal wave length
            thermalWaveLength =
sqrt(hplanck*hplanck/twoPi/kboltzmann/temperature/mass*6.02214e+26)*1.0e+1
0;
```

```
                double probScale;
                //calculate the probScale
                probScale =
thermalWaveLength*thermalWaveLength*thermalWaveLength*moleculeContain
er.size()/system_x/system_y/system_z;

                exchange(choice);//exchange the choosed particle with the last
particle

                double deleteEnergy;
                if(checkTime == 1)
                    time(&start);
                //check the Lennard-Jones potential and real space energy
                deleteEnergy = singleEnergy();
                if(checkTime == 1)
                {
                    time(&end);
                    deleteReal += end - start;
                }
                //check the time
                if(checkTime == 1)
                    time(&start);
                //check the energy of the choosed molecule in the system
                deleteEnergy = deleteEnergy - self(moleculeContainer[last]) +
ewaldDelete() - moleculeContainer[last].intra(kappa, wallmin);
                //check the time comsumption of reciprocal part
                if(checkTime == 1)
                {
                    time(&end);
                    deleteReciprocal += end - start;
                }
                double acceptFactor;
                //calculate the accept factor
                acceptFactor = idratio*probScale*exp(-(chemPotential -
deleteEnergy)/KbT);
                //accept the delete trial if the random number is less than accept factor
                if(ran() < acceptFactor)
                {
                    //renew ewald
                    ewaldDeleteExe(choice);
                    //output the energy of the deleted molecule and molecule number
```

```
                    if(acceptFactor > 1)
                        cout<<"Delete Accepted(energy): "<<deleteEnergy<<"
moleclue number: "<<moleculeContainer.size()<<endl;
                    else
                        cout<<"Delete Accepted(random number):
"<<deleteEnergy<<" molecule number: "<<moleculeContainer.size()<<endl;
                }
                //exchange the choosed molecule back if not accept the delete trial
                else
                {
                    ewaldDeleteUnexe(choice);
                }
                //check the time comsumption of delete trial
                if(checkTime == 1)
                {
                    time(&endTotal);
                    deleteTotal += endTotal - startTotal;
                }
            }

            //insert trial
            void insertion()
            {
                Molecule insertMolecule;
                int choice;
                time_t start, end, startTotal, endTotal;
                //check the time
                if(checkTime == 1)
                    time(&startTotal);

                //randomly choose a molecule from model container to insert into the
system
                choice = rand()%modelContainer.size();
                insertMolecule = modelContainer[choice];

                //random move the choosed molecule from center of the system
                double px, py, pz;
                px = (2*ran() - 1) * system_x/2.0;
                py = (2*ran() - 1) * system_y/2.0;
                pz = (2*ran() - 1) * system_z/2.0;
```

```
                insertMolecule.move(px, py, pz, twoPi, ewaldTempX, ewaldTempY,
ewaldTempZ);
                //rotate the choosed molecule
                insertMolecule.rotate(twoPi, ewaldTempX, ewaldTempY,
ewaldTempZ);
                //push the molecule into molecule container
                moleculeContainer.push_back(insertMolecule);

                int last;
                last = moleculeContainer.size() - 1;

                double mass;
                mass = insertMolecule.getMass();
                double thermalWaveLength;
                //calculate the thermal wave length
                thermalWaveLength =
sqrt(hplanck*hplanck/twoPi/kboltzmann/temperature/mass*6.02214e+26)*1.0e+1
0;
                double probScale;
                //calculate the probScale
                probScale =
system_x*system_y*system_z/thermalWaveLength/thermalWaveLength/thermal
WaveLength/moleculeContainer.size();

                double insertEnergy;
                //check the start time
                if(checkTime == 1)
                    time(&start);
                //calculate the Lennard-Jones potential and real space energy
                insertEnergy = singleEnergy();
                //check the time comsuption of real space part in insert trial
                if(checkTime == 1)
                {
                    time(&end);
                    insertReal += end - start;
                }

                //check the reciprocal part time
                if(checkTime == 1)
                    time(&start);
                //calculate the energy of the choosed molecule
```

```
                insertEnergy = insertEnergy - self(moleculeContainer[last]) +
ewaldInsert() - moleculeContainer[last].intra(kappa, wallmin);
                //check the time comsuption of reciprocal part in insert trial
                if(checkTime == 1)
                {
                    time(&end);
                    insertReciprocal += end - start;
                }
                double acceptFactor;
                //calculate the accept factor
                acceptFactor = (1./idratio)*probScale*exp(-(insertEnergy -
chemPotential)/KbT);

                //accept the trial if the random number is less than the accept factor
                if(ran() < acceptFactor)
                {
                    //renew the ewald
                    ewaldInsertExe();
                    if(acceptFactor > 1)
                        cout<<"Insert Accepted (energy): "<<insertEnergy<<"
 molecule number: "<<moleculeContainer.size()<<endl;
                    else
                        cout<<"Insert Accepted (random number):
"<<insertEnergy<<" molecule number: "<<moleculeContainer.size()<<endl;
                }
                //delete the choosed molecule from the molecule container if not
accept the trial
                else
                    ewaldInsertUnexe();

                //check time comsuption of insert trial
                if(checkTime == 1)
                {
                    time(&endTotal);
                    insertTotal += endTotal - startTotal;
                }
            }

            //self energy in ewald energy calculation
            double self(Molecule m)
            {
```

```
        double energySelf;

        energySelf = m.chargeSqr();

        energySelf = energySelf * kappa / wallmin * rSqrtPi;

        return energySelf;
    }

    //set up the k in the ewald energy calculation
    void ksetup()
    {
        int xmax, ymax, zmax;
        //set the system to mesh
        xmax = int(kMax * (ewaldTempX/wallmin));
        ymax = int(kMax * (ewaldTempY/wallmin));
        zmax = int(kMax * (ewaldTempZ/wallmin));

        double rksq, rksqmax;
        rksqmax = double(ksqMax);
        //not calculate all possibilities for the symmetry to reduce the
calculation
        for(int nx = 0; nx <= xmax; nx++)
            for(int ny = -ymax; ny <= ymax; ny++)
                for(int nz = -zmax; nz <= zmax; nz++)
                {
                    if(nx == 0)
                    {
                        if(ny < 0) continue;
                        if(ny ==0 && nz <= 0) continue;
                    }
                    rksq =
sqr(double(nx)/ewaldTempX)+sqr(double(ny)/ewaldTempY)+sqr(double(nz)/ewa
ldTempZ);
                    if(rksq*sqr(wallmin) <= rksqmax)
                    {
                        if(ihx.size() > mvec)
                            cout<<"gmothers"<<endl;
                        //push ihx, ihy and ihz into the container
                        ihx.push_back(nx);
                        ihy.push_back(ny);
```

```
                                ihz.push_back(nz);
                                double temp;
                                temp =
1.0/rksq*exp(-piSqr*rksq/sqr(kappa/wallmin));
                                fhold.push_back(temp);
                            }
                        }
                }

                //reset the system size if has the wall because the wall has thick
                void wallThick()
                {
                    if(haveWall)
                    {
                        if(!boundaryX)
                            ewaldTempX = system_x + thickX;
                        else
                            ewaldTempX = system_x;
                        if(!boundaryY)
                            ewaldTempY = system_y + thickY;
                        else
                            ewaldTempY = system_y;
                        if(!boundaryZ)
                            ewaldTempZ = system_z + thickZ;
                        else
                            ewaldTempZ = system_z;
                    }
                    else
                    {
                        ewaldTempX = system_x;
                        ewaldTempY = system_y;
                        ewaldTempZ = system_z;
                    }
                }

                //calculate ewald energy
                void ewald()
                {
                    double ndotr;

                    //initialize the sum of sin and cos to be zero
```

```
            for(int i = 0; i < ihx.size(); i++)
            {
                ssum.push_back(0);
                csum.push_back(0);
            }

            //check all points in the mesh
            for(int i = 0; i < ihx.size(); i++)
                //check all the molecule in the system
                for(int j = 0; j < moleculeContainer.size(); j++)
                {
                    vector<double> c, rxt, ryt, rzt;
                    c = moleculeContainer[j].getChargeContainer();
                    //get rx, ry and rz from the container
                    rxt = moleculeContainer[j].getRX();
                    ryt = moleculeContainer[j].getRY();
                    rzt = moleculeContainer[j].getRZ();
                    //calculate the sum of sin and cos for all the atoms in the
molecule
                    for(int k = 0; k < c.size(); k++)
                    {
                        ndotr = ihx[i]*rxt[k] + ihy[i]*ryt[k] + ihz[i]*rzt[k];
                        ssum[i] += c[k]*sin(ndotr);
                        csum[i] += c[k]*cos(ndotr);
                    }
                }

            //get a copy of the sum of sin and cos
            for(int i = 0; i < ihx.size(); i++)
            {
                ssum2.push_back(ssum[i]);
                csum2.push_back(csum[i]);
            }
        }

        //calculate ewald energy in delete trial
        double ewaldDelete()
        {
            int last;
            last = moleculeContainer.size() - 1;
```

```
            vector<double> s, c;
            moleculeContainer[last].singleEwald(s, c, ihx, ihy, ihz);

            //minus the sum of the sin and cos contribution from the last molecule
in the system
            for(int i = 0; i < ihx.size(); i++)
            {
                ssum2[i] -= s[i];
                csum2[i] -= c[i];
            }

            double energyEwaldDelete = 0;

            //check the energy difference
            for(int i = 0; i < ihx.size(); i++)
            {
                energyEwaldDelete += fhold[i]*(csum[i]*csum[i] +
ssum[i]*ssum[i] - csum2[i]*csum2[i] - ssum2[i]*ssum2[i]);
            }
            energyEwaldDelete =
2.0*energyEwaldDelete/(twoPi*ewaldTempX*ewaldTempY*ewaldTempZ);
            return energyEwaldDelete;
        }

        //calculate ewald energy in insert trial
        double ewaldInsert()
        {
            int last;
            last = moleculeContainer.size() - 1;

            vector<double> s, c;
            moleculeContainer[last].singleEwald(s, c, ihx, ihy, ihz);
            //plus the sum of sin and cos contribution from the choosed molecule
            for(int i = 0; i < ihx.size(); i++)
            {
                ssum2[i] += s[i];
                csum2[i] += c[i];
            }

            double energyEwaldInsert = 0;
```

```
            //calculate the energy difference
            for(int i = 0; i < ihx.size(); i++)
            {
                energyEwaldInsert += fhold[i]*(csum2[i]*csum2[i] +
ssum2[i]*ssum2[i] - csum[i]*csum[i] - ssum[i]*ssum[i]);
            }

            energyEwaldInsert =
2.0*energyEwaldInsert/(twoPi*ewaldTempX*ewaldTempY*ewaldTempZ);
            return energyEwaldInsert;
        }

        //calculate ewald energy in movement trial
        void ewaldMove(bool isBefore, double &energyEwaldMove)
        {
            int last;
            last = moleculeContainer.size() - 1;

            energyEwaldMove = 0;

            vector<double> s, c;

            //minus the sum of sin and cos contribution from the choosed
molecule before movement
            if(isBefore == true)
            {
                moleculeContainer[last].singleEwald(s, c, ihx, ihy, ihz);
                for(int i = 0; i < ihx.size(); i++)
                {
                    ssum2[i] -= s[i];
                    csum2[i] -= c[i];
                }
            }
            //plus the sum of sin and cos contribution from the choosed molecule
after movement
            else
            {
                moleculeContainer[last].singleEwald(s, c, ihx, ihy, ihz);
                for(int i = 0; i < ihx.size(); i++)
                {
                    ssum2[i] += s[i];
```

```
                    csum2[i] += c[i];
                }

                //calculate the energy difference
                for(int i = 0; i < ihx.size(); i++)
                {
                    energyEwaldMove += fhold[i]*(csum2[i]*csum2[i] +
ssum2[i]*ssum2[i] - csum[i]*csum[i] - ssum[i]*ssum[i]);
                }

                energyEwaldMove =
2.0*energyEwaldMove/(twoPi*ewaldTempX*ewaldTempY*ewaldTempZ);
            }
        }

        //renew information after the movement trial is accepted
        void ewaldMoveExe(int choice)
        {
            //renew the sum of sin and cos
            sinCosSumRenew();
            //exchange the choosed molecule to initial position in the sequence
            exchange(choice);
        }

        //renew information if the movement trial is not accepted
        void ewaldMoveUnexe(int choice)
        {
            //renew the sum of sin and cos
            sinCosSumRestore();
            //exchange the choosed molecule to initial position in the sequence
            exchange(choice);
        }

        //renew information after the delete trial is accepted
        void ewaldDeleteExe(int choice)
        {
            //renew the sum of sin and cos
            sinCosSumRenew();
            //exchange the choosed molecule to initial position in the sequence
            exchange(choice);
            //delete the molecule from the molecule container
```

```
            moleculeContainer.erase(moleculeContainer.begin() + choice);

        }

        //renew information if the delete trial is not accepted
        void ewaldInsertExe()
        {
            //renew the sum of sin and cos
            sinCosSumRenew();
        }

        //renew information after the insert trial is accepted
        void ewaldInsertUnexe()
        {
            //renew the sum of sin and cos
            sinCosSumRestore();
            //delete the last moleucle in the molecule container
            moleculeContainer.pop_back();
        }


        //renew information if the insert trial is not accepted
        void ewaldDeleteUnexe(int choice)
        {
            //renew the sum of sin and cos
            sinCosSumRestore();
            //exchange the choosed molecule to the initial position in the
sequence
            exchange(choice);
        }

        //ewald energy for full energy calculation
        double ewaldFull()
        {
            double ewaldFullTemp;
            ewaldFullTemp = 0.0;

            for(int i = 0; i < ihx.size(); i++)
                ewaldFullTemp +=
fhold[i]*(csum[i]*csum[i]+ssum[i]*ssum[i]);
            ewaldFullTemp =
```

```
2.0*ewaldFullTemp/(twoPi*ewaldTempX*ewaldTempY*ewaldTempZ);

            return ewaldFullTemp;
        }

        //calculate the total energy in the system
        double energyFull()
        {
            double energyFullTemp;
            energyFullTemp = 0.0;
            int last;
            last = moleculeContainer.size() - 1;

            //check total energy of the molecules in the system
            for(int i = 0; i < moleculeContainer.size(); i++)
            {
                exchange(i);
                energyFullTemp += singleEnergy() -
self(moleculeContainer[last]) - moleculeContainer[last].intra(kappa, wallmin);
                exchange(i);
            }

            energyFullTemp += ewaldFull();

            //consider the effect of double counting
            return energyFullTemp/2.0;
        }

        //return the average energy
        double averageFullEnergy()
        {
            return energyFull()/moleculeContainer.size();
        }

        //restore the sum of sin and cos
        void sinCosSumRestore()
        {
            for(int i = 0; i < ssum.size(); i++)
            {
                ssum2[i] = ssum[i];
                csum2[i] = csum[i];
            }
```

```
        }

        //renew the sum of sin and cos
        void sinCosSumRenew()
        {
            for(int i = 0; i < ssum.size(); i++)
            {
                ssum[i] = ssum2[i];
                csum[i] = csum2[i];
            }
        }

        //calculate the Lennard-Jones potential and real space energy in the ewald energy calculation
        double singleEnergy()
        {
            int last;
            last = moleculeContainer.size() - 1;
            double energy = 0;
            for(int i = 0; i < moleculeContainer.size() - 1; i++)
            {
                energy += moleculeContainer[last].interEnergy(moleculeContainer[i], kappa, wallmin, system_x, system_y, system_z, haveWall, boundaryX, boundaryY, boundaryZ);
            }
            //consider the wall effect
            if(haveWall)
                energy += moleculeContainer[last].wallInteraction(system_x, system_y, system_z, twoPi, boundaryX, thickX, boundaryY, thickY, boundaryZ, thickZ, sigWallX, epsWallX, sigWallY, epsWallY, sigWallZ, epsWallZ, rhoX, rhoY, rhoZ);
                    return energy;
        }

        //real space energy in ewald
        double realspace(double r)
        {
            double a1 = 0.254829592;
            double a2 = -0.284496736;
            double a3 = 1.421413741;
            double a4 = -1.453152027;
```

```
	double a5 = 1.061405429;
	double pew = 0.3275911;

	double t, xsq, tp, real, rkap, erfc;

	rkap = (kappa/wallmin)*r;
	t = 1.0/(1.0 + pew * rkap);
	xsq = rkap * rkap;
	tp = t * ( a1 + t * ( a2 + t * ( a3 + t * ( a4 + t * a5 ) ) ) );
	erfc = tp*exp(-xsq);
	real = erfc/r;

	return real;
}


//exchange the choosed molecule with the last molecule in the system
void exchange(int i)//exchange choosed particle with last particle
{
	Molecule temp;
	int last;
	last = moleculeContainer.size() - 1;
	temp = moleculeContainer[i];
	moleculeContainer[i] = moleculeContainer[last];
	moleculeContainer[last] = temp;
}

//return a number within [0,1)
double ran()
{
	return double(rand())/(double(RAND_MAX) + double(1));
}

//return the square of the number
int sqr(int n) {return n*n;}

//return the square of the real number
double sqr(double n) {return n*n;}

//analysis the time spend of each part
void timeAnalysis()
{
```

```
                cout<<"Insertion part spend "<<insertTotal<<" seconds:"<<endl;
                cout<<"      Real part: "<<insertReal<<" seconds,
"<<double(insertReal)/insertTotal*100<<"%"<<endl;
                cout<<"      Reciprocal part: "<<insertReciprocal<<" seconds,
"<<double(insertReciprocal)/insertTotal*100<<"%"<<endl;
                cout<<"Movement part spend "<<moveTotal<<" seconds:"<<endl;
                cout<<"      Real part: "<<moveReal<<" seconds,
"<<double(moveReal)/moveTotal*100<<"%"<<endl;
                cout<<"      Reciprocal part: "<<moveReciprocal<<" seconds,
"<<double(moveReciprocal)/moveTotal*100<<"%"<<endl;
                cout<<"Deletion part spend "<<deleteTotal<<" seconds:"<<endl;
                cout<<"      Real part: "<<deleteReal<<" seconds,
"<<double(deleteReal)/deleteTotal*100<<"%"<<endl;
                cout<<"      Reciprocal part: "<<deleteReciprocal<<" seconds,
"<<double(deleteReciprocal)/deleteTotal*100<<"%"<<endl;
                cout<<"Totally, real part spend "<<double(insertReal + moveReal +
deleteReal)/(insertTotal + moveTotal + deleteTotal)*100<<"%, reciprocal part
spend "<<double(insertReciprocal + moveReciprocal +
deleteReciprocal)/(insertTotal + moveTotal + deleteTotal)*100<<"%."<<endl;
        }

        //plot the molecule number vs cycle for each molecule
        void moleculeNumberPlot(int cycle, ofstream &file)
        {
            int *ptr;
            ptr = new int [modelContainer.size()];

            //initial the array to be zero
            for(int i = 0; i < modelContainer.size(); i++)
                ptr[i] = 0;

            //plot the molecule number vs cycle
            for(int i = 0; i < modelContainer.size(); i++)
                for(int j = 0; j < moleculeContainer.size(); j++)
                    if(modelContainer[i].getName() ==
moleculeContainer[j].getName())
                        ptr[i]++;

            file<<setw(10)<<cycle;
            for(int i = 0; i < modelContainer.size(); i++)
                file<<setw(10)<<ptr[i];
```

```
            file<<endl;

            delete [] ptr;
        }

        //plot the gr function
        void gr(ofstream &file)
        {
            int rowNum, columnNum;
            int frontNum, backNum;
            static int countGr = 0;
            //if the system include three molecule, 0, 1, 2, 3, the column is 00, 01,
02, 03, 11, 12, 13, 22, 23, 33
            for(int i = 0; i < moleculeContainer.size()-1; i++)
                for(int j = i+1; j < moleculeContainer.size(); j++)
                {
                    //find column number
                    for(int k = 0; k < modelContainer.size(); k++)
                    {
                        if(moleculeContainer[i].getName() ==
modelContainer[k].getName())
                            frontNum = k;
                        if(moleculeContainer[j].getName() ==
modelContainer[k].getName())
                            backNum = k;
                    }
                    columnNum = 0;
                   for(int k = modelContainer.size(); k > modelContainer.size()
- frontNum; k--)
                    {
                        columnNum += k;
                    }
                    columnNum = columnNum + backNum - frontNum;
                    //find row number
                    rowNum =int(distance(moleculeContainer[i],
moleculeContainer[j], system_x, system_y, system_z)*grResolution);
                    //renew gr array
                    grArray[rowNum][columnNum] += 2.0;
                }
            grCount++;
            for(int i = 0; i < modelContainer.size(); i++)
```

```
                    for(int j = 0; j < moleculeContainer.size(); j++)
                        if(modelContainer[i].getName() ==
moleculeContainer[j].getName())
                            grCountNum[i]++;
                double deltV, deltD, radii;
                double tmp;
                for(int i = 0; i < grRow; i++)
                {
                    radii = double(i)/grResolution;
                    file<<setprecision(3)<<fixed<<setw(5)<<radii;
                    for(int j = 0; j < grColumn; j++)
                    {
                        deltD = double(1)/grResolution;
                        deltV = 4*3.1415926*radii*radii*deltD;
                        tmp =
deltV*grCount*grCountNum[0]/grCount*grCountNum[0]/grCount/system_x/syst
em_y/system_z;
                        file<<setprecision(4)<<fixed<<"
"<<setw(8)<<grArray[i][j]/tmp;
                    }
                    file<<endl;
                }

            }

            //return the distance between two molecules
            double distance(Molecule m1, Molecule m2, double rangeX, double
rangeY, double rangeZ)
            {
                return m1.centralDistance(m2, rangeX, rangeY, rangeZ);
            }

            //generate the pdb file
            void generatePdb(){
                ofstream file;
                file.open("output.pdb");
                for(int i = 0; i < moleculeContainer.size(); i++)
                {
                    moleculeContainer[i].moleculePdb(file, i);
                }
                file.close();
            }
```

```
//Accessor
//return if has the wall
bool getHaveWall() {return haveWall;}
//get the system size at x direction
double getSystemX() {return system_x;}
//get the system size at y direction
double getSystemY() {return system_y;}
//get the system size at z direction
double getSystemZ() {return system_z;}
//get the minimum size
double getWallmin() {return wallmin;}
//get the temperature
double getTemperature() {return temperature;}
//get the movint step
double getMoveStep() {return moveStep;}
//get the rotate step
double getRotateStep() {return rotateStep;}
//get how many particles will be deleted
int getLessParticle() {return lessParticle;}
//return if the system only execute the movement trial
bool getNotOnlyMove() {return notOnlyMove;}
int getModGz() {return modGz;}
int getModGr() {return modGr;}
//get the thick of the wall
double getThickOfWall() {return thickOfWall;}
//get the factor of the wall
double getFactorWall() {return factorWall;}
//return if consider the boundary condition at x direction
bool getHaveBoundaryX() {return boundaryX;}
//return if consider the boundary condition at y direction
bool getHaveBoundaryY() {return boundaryY;}
//return if consider the boundary condition at z direction
bool getHaveBoundaryZ() {return boundaryZ;}
//get kMax
int getkMax() {return kMax;}
//get ksqMax
int getksqMax() {return ksqMax;}
//get kappa
double getKappa() {return kappa;}
int getSlide() {return slide;}
```

```
        //get the molecule number in the molecule container
        int getMoleculeNumber() {return moleculeContainer.size();}
        //get the molecule number in the model container
        int getModelNumber() {return modelContainer.size();}
        //get the ratio of insert to delete
        int getIdratio() {return idratio;}
};

/*Molecule.h*/
/*Descript the molecule that can be inserted into the system*/
#ifndef MOLECULE_H
#define MOLECULE_H
#include <iostream>
#include <fstream>
#include <vector>
#include <string>
#include <iomanip>
#include <cmath>
#include "Atom.h"
using namespace std;

class Molecule
{
    private: string name;//represent molecule name
        int siteNumber;//define how many atoms in the molecule
        double mass;//represent the total mass of the molecule
        vector<Atom> atomContainer;//build a atom container to save the atoms
in the molecule
        vector<double> chargeContainer, rx, ry, rz;//build the containers to save
the charge, rx, ry and rz in the molecule, and rx, ry and rz the mediate value when do
the ewald energy calculation
        bool fixing;//represent if the molecule can move, 1 fix the position in the
system, 0 the molecule can move, by which can insert the block that do not do the
insert, delete and movement trial, such as carbon nanotube

    public:
        //Constructor
        Molecule()
        {
        }
```

```
        //read the parameters and atoms of the molecule from the file
        Molecule(int n, ifstream &inputFile, double twopi, double system_x,
double system_y, double system_z)
        {
            siteNumber = n;
            read(inputFile,twopi, system_x, system_y, system_z);
        }

        //Deconstructor
        ~Molecule()
        {
        }

        //Mutator
        //set the atom number in the molecule
        void setSiteNumber(int n)
        {
            siteNumber = n;
        }

        //set the moleclule name
        void setName(string n)
        {
            name = n;
        }

        //save the molecule into the file
        void save(ofstream &file)
        {
            //save the name, siteNumber, mass and fixing to the file

file<<fixed<<setprecision(4)<<setw(10)<<name<<setw(10)<<siteNumber<<setw
(10)<<mass<<setw(10)<<fixing<<endl;
            //save the atom to the file
            for(int i = 0; i < siteNumber; i++)
            {
                atomContainer[i].save(file);
            }
        }

        //read the parameters and atoms of the molecule from the file
```

```
        void read(ifstream &file, double twopi, double wallx, double wally,
double wallz)
        {
            Atom atom;
            //read the name, siteNumber, mass and fixing
            file>>name>>siteNumber>>mass>>fixing;
            //read the atoms and push into the atomContainer
            for(int i = 0; i < siteNumber; i++)
            {
                atom.read(file);
                atomContainer.push_back(atom);
            }

            //give the error information if the siteNumber does match the atom
number in the molecule
            if(atomContainer.size() != siteNumber)
            {
                cout<<"Error is found in reading molecule!"<<endl;
            }

            //initialize the rx, ry and rz
            setR(twopi, wallx, wally, wallz);
        }

        //set up the rx, ry and rz which is the molecule parameters when calculate
the ewald potential
        void setR(double twopi, double wallx, double wally, double wallz)
        {
            //clear the containers for charge, rx, ry and rz
            chargeContainer.clear();
            rx.clear();
            ry.clear();
            rz.clear();
            double tempx, tempy, tempz;
            //push the charge of each atom in the molecule into the
chargeContainer
            //calculate the rx, ry and rz value and push into the correspond
container
            for(int i = 0; i < siteNumber; i++)
            {
                tempx = twopi/wallx*atomContainer[i].getX();
```

```
            tempy = twopi/wallx*atomContainer[i].getY();
            tempz = twopi/wally*atomContainer[i].getZ();
            chargeContainer.push_back(atomContainer[i].getCharge());
            rx.push_back(tempx);
            ry.push_back(tempy);
            rz.push_back(tempz);
        }
    }


    //define the equal that can copy the molecule information to the other
molecule
    void operator= (const Molecule &right)
    {
        name = right.name;
        siteNumber = right.siteNumber;
        mass = right.mass;
        fixing = right.fixing;
        atomContainer = right.atomContainer;
        chargeContainer = right.chargeContainer;
        rx = right.rx;
        ry = right.ry;
        rz = right.rz;
    }

    //shift the molecule a little bit
    //it is useful in the movement trial
    void shift(double step, double twopi, double wallx, double wally, double
wallz)
    {
        double dx, dy, dz;

        //generate the shift step randomly
        dx = (ran() - 0.5)*step;
        dy = (ran() - 0.5)*step;
        dz = (ran() - 0.5)*step;

        //shift every atom in the molecule
        for(int i = 0; i < atomContainer.size(); i++)
        {
            atomContainer[i].shift(dx, dy, dz);
        }
```

```
            //renew the rx, ry and rz value
            setR(twopi, wallx, wally, wallz);
        }

        //move the molecule
        //it is useful in the insert trial
        void move(double dx, double dy, double dz, double twopi, double wallx,
double wally, double wallz)
        {
            //move every atom in the molecule
            for(int i = 0; i < atomContainer.size(); i++)
            {
                atomContainer[i].shift(dx, dy, dz);
            }

            //renew the rx, ry and rz value
            setR(twopi, wallx, wally, wallz);
        }
        //rotate the moleule a little bit around the firt atom in the molecule
        //it is useful in the movement trial
        void tinyRotate(double twoPi, double step, double wallx, double wally,
double wallz) //tiny quaternion rotate
        {
            double gammar, betar, alphar;
            Atom *temp, *temp2;
            temp = new Atom[atomContainer.size()];
            temp2 = new Atom[atomContainer.size()];

            //set up a pointer to build the rotate matrix
            double **rotate;
            rotate = new double *[3];

            for(int i = 0; i < 3; i++)
                rotate[i] = new double[3];

            gammar = twoPi*(ran()-0.5)*step;
            betar = twoPi*(ran()-0.5)*step;
            alphar = twoPi*(ran()-0.5)*step;
            rotate[0][0] = cos(gammar)*cos(betar)*cos(alphar) -
sin(gammar)*sin(alphar);
```

```
                rotate[0][1] = cos(gammar)*cos(betar)*sin(alphar) +
sin(gammar)*cos(alphar);
                rotate[0][2] = -cos(gammar)*sin(betar);
                rotate[1][0] = -sin(gammar)*cos(betar)*cos(alphar) -
cos(gammar)*sin(alphar);
                rotate[1][1] = -sin(gammar)*cos(betar)*sin(alphar) +
cos(gammar)*cos(alphar);
                rotate[1][2] = sin(gammar)*sin(betar);
                rotate[2][0] = sin(betar)*cos(alphar);
                rotate[2][1] = sin(betar)*sin(alphar);
                rotate[2][2] = cos(betar);


                //move the molecule to the center of the system
                for(int i = 1; i < atomContainer.size(); i++)
                    temp[i] = atomContainer[i] - atomContainer[0];

                //rotate the molecule
                for(int i = 1; i < atomContainer.size(); i++)
                    temp2[i] = temp[i].rotate(rotate, 3);

                //move the center of the molecule back to the initial position
                for(int i = 1; i < atomContainer.size(); i++)
                    atomContainer[i] = temp2[i] + atomContainer[0];

                //delete the pointer
                for(int i = 0; i < 3; i++)
                    delete [] rotate[i];

                delete [] rotate;
                delete [] temp;
                delete [] temp2;

                //renew the rx, ry and rz value
                setR(twoPi, wallx, wally, wallz);
            }

            //randomly rotate the molecule around the first atom of the molecule
            //it is usefule in the insert trial
            void rotate(double twopi, double wallx, double wally, double
wallz)//random quaternion rotate
            {
```

```
double chi, mu, S1, iota, zeta, S2;
Atom *temp, *temp2;
temp = new Atom[atomContainer.size()];
temp2 = new Atom[atomContainer.size()];

//build the rotate matrix
double **rotate;
    rotate = new double *[3];
for(int i = 0; i < 3; i++)
    rotate[i] = new double[3];

chi = 2 * ran() - 1;
mu = 2 * ran() - 1;
S1 = chi * chi + mu * mu;
while(S1 >= 1)
{
    chi = 2 * ran() - 1;
    mu = 2 * ran() - 1;
    S1 = chi * chi + mu * mu;
}

iota = 2 * ran() - 1;
zeta = 2 * ran() - 1;
S2 = iota * iota + zeta * zeta;
while(S2 >= 1)
{
    iota = 2 * ran() - 1;
    zeta = 2 * ran() - 1;
    S2 = iota * iota + zeta * zeta;
}

iota = iota * sqrt((1-S1)/S2);
zeta = zeta * sqrt((1-S1)/S2);

rotate[0][0] = chi*chi - mu*mu - iota*iota + zeta*zeta;
rotate[0][1] = 2.0*((chi*mu) - (iota*zeta));
rotate[0][2] = 2.0*((iota*chi) + (mu*zeta));
rotate[1][0] = 2.0*((chi*mu) + (iota*zeta));
rotate[1][1] = mu*mu - iota*iota - chi*chi + zeta*zeta;
rotate[1][2] = 2.0*((mu*iota) - (chi*zeta));
rotate[2][0] = 2.0*((iota*chi) - (mu*zeta));
```

```
            rotate[2][1] = 2.0*((mu*iota) + (chi*zeta));
            rotate[2][2] = iota*iota - chi*chi - mu*mu + zeta*zeta;

            //move the molecule to the center of the system
            for(int i = 1; i < atomContainer.size(); i++)
                temp[i] = atomContainer[i] - atomContainer[0];

            //rotate the molecule
            for(int i = 1; i < atomContainer.size(); i++)
                temp2[i] = temp[i].rotate(rotate, 3);

            //move the center of the molecule to the intital position
            for(int i = 1; i < atomContainer.size(); i++)
                atomContainer[i] = temp2[i] + atomContainer[0];

            //delete the pointer
            for(int i = 0; i < 3; i++)
                delete [] rotate[i];

            delete [] rotate;
            delete [] temp;
            delete [] temp2;

            //renew the rx, ry and rz value
            setR(twopi, wallx, wally, wallz);
        }

        //move the molecule into the system if the molecule is out of range after movement trial
        void align(double bx, double by, double bz, double twopi, double wallx, double wally, double wallz)
        {
            int xIsOver = 0, yIsOver = 0, zIsOver = 0;

            //tell if the first atom is out of the system, 1 is out of the right boundary, -1 is out of the left boundary, 0 is not out of the system
            if(atomContainer[0].getX() > bx/2.0) xIsOver = 1;
            if(atomContainer[0].getX() < -bx/2.0) xIsOver = -1;

            if(atomContainer[0].getY() > by/2.0) yIsOver = 1;
            if(atomContainer[0].getY() < -by/2.0) yIsOver = -1;
```

```
            if(atomContainer[0].getZ() > bz/2.0) zIsOver = 1;
            if(atomContainer[0].getZ() < -bz/2.0) zIsOver = -1;

            //align every atom in the molecule
            for(int i = 0; i < atomContainer.size(); i++)
                atomContainer[i].align(bx,xIsOver,by,yIsOver,bz,zIsOver);

            //renew the rx, ry and rz value
            setR(twopi, wallx, wally, wallz);
        }

        //tell if the molecule is out of the sytem if the system include the wall
        bool outsideTell(double bx, double by, double bz, bool wall, bool tx, bool
ty, bool tz)
        {
            //check the molecule if the system has the wall
            if(wall)
            {
            //check the molecule if the system has the wall at x direction
            if(!tx)
            {
                if(fabs(atomContainer[0].getX()) > bx/2.0)
                    return true;
            }
            //check the molecule if the system has the wall at y direction
            if(!ty)
            {
                if(fabs(atomContainer[0].getY()) > by/2.0)
                    return true;
            }
            //check the molecule if the system has the wall at z direction
            if(!tz)
            {
                if(fabs(atomContainer[0].getZ()) > bz/2.0)
                    return true;
            }
            return false;
            }
            else
                return false;
        }
```

```
        //return the sum of the sqrare value of the charge of the atoms in the
molecule
        double chargeSqr()
        {
            double charge, chargesqr = 0;

            for(int i = 0; i < atomContainer.size(); i++)
            {
                charge = atomContainer[i].getCharge();
                chargesqr += charge*charge;
            }

            return chargesqr;
        }

        //calculate the sum(q*cos(kr)) and sum(q*sin(kr))
        void singleEwald(vector<double> &s, vector<double> &c, vector<int>
ihx, vector<int> ihy, vector<int> ihz)
        {
            double dot;
            s.clear();
            c.clear();
            //initialize sum of cos and sin to be zero
            for(int i = 0; i < ihx.size(); i ++)
            {
                s.push_back(0);
                c.push_back(0);
            }
            //calculate sum of cos and sin for every atom in the molecule
            for(int i = 0; i < siteNumber; i++)
            {
                //check all the point on the mesh
                for(int j = 0; j < ihx.size(); j++)
                {
                    //dot product kr
                    dot =   ihx[j]*rx[i] + ihy[j]*ry[i] + ihz[j]*rz[i];
                    //calculate the sum of sin
                    s[j] += chargeContainer[i]*sin(dot);
                    //calculate the sum of cos
                    c[j] += chargeContainer[i]*cos(dot);
                }
            }
```

```
        }

        //calculate the energy between molecules
        double interEnergy(Molecule &m, double kappa, double wallmin, double
wallx, double wally, double wallz, bool haveWall, bool boundaryX, bool
boundaryY, bool boundaryZ)
        {
            double d;
            Atom atomA, atomB;
            double energy = 0;
            double tempPotential;

            //calculate the energy between the atoms of first molecule and the
atoms of the second molecule
            for(int i = 0; i < atomContainer.size(); i++)
            {
                atomA = atomContainer[i];
                for(int j = 0; j < m.atomContainer.size(); j++)
                {
                    atomB = m.atomContainer[j];
                    //calculate the distance between two atoms
                    d = interAtomDistance(atomA, atomB, wallx, wally, wallz,
haveWall, boundaryX, boundaryY, boundaryZ);
                    //calculate the Lennard-Jones potential
                    if((atomA.getEps() == 0) || (atomB.getEps() == 0))
                        tempPotential = 0;
                    else
                        tempPotential = potentialLJ(d, atomA, atomB);
                    //calculate the real space part of ewald energy
                    energy += tempPotential +
atomA.getCharge()*atomB.getCharge()*realspace(d, kappa, wallmin);
                }
            }
           return energy;
        }

        //calculate the Lennard-Jones potential between two atoms
        double potentialLJ(double r, Atom a, Atom b)
        {
            double sigma, epsilon, energy;
```

```
            sigma = (a.getSig() + b.getSig())/2.0;
            epsilon = sqrt(a.getEps()*b.getEps());

            double ep4, exp1;
            ep4 = 4.0*epsilon;
            exp1 = sixTime(sigma/r);
            energy = ep4*exp1*(exp1 - 1.0);

            return energy;
        }

        //return the six power value
            double sixTime(double x)
        {
            return x*x*x*x*x*x;
        }

        //calculate the distance between two atoms
        double interAtomDistance(Atom a, Atom b, double wallx, double wally,
double wallz, bool haveWall, bool boundaryX, bool boundaryY, bool boundaryZ)
        {
            double temp, tempx, tempy, tempz;

            //do not consider the boundary condition when there is wall in this
direction
            if(haveWall)
            {
                if(!boundaryX)
                {
                    tempx = a.getX() - b.getX();
                    tempx = tempx*tempx;
                }
                else
                {
                    tempx = a.getX() - b.getX();
                    tempx = tempx -
wallx*(int(tempx*2/wallx)-int(tempx/wallx));
                    tempx = tempx*tempx;
                }
                if(!boundaryY)
                {
```

```
                    tempy = a.getY() - b.getY();
                    tempy = tempy*tempy;
                }
                else
                {
                    tempy = a.getY() - b.getY();
                    tempy = tempy -
wally*(int(tempy*2/wally)-int(tempy/wally));
                    tempy = tempy*tempy;
                }
                if(!boundaryZ)
                {
                    tempz = a.getZ() - b.getZ();
                    tempz = tempz*tempz;
                }
                else
                {
                    tempz = a.getZ() - b.getZ();
                    tempz = tempz -
wallz*(int(tempz*2/wallz)-int(tempz/wallz));
                    tempz = tempz*tempz;
                }
            }
            //consider boundary condition at all direction
            else
            {
                tempx = a.getX() - b.getX();
                tempx = tempx - wallx*(int(tempx*2/wallx)-int(tempx/wallx));
                tempx = tempx*tempx;
                tempy = a.getY() - b.getY();
                tempy = tempy - wally*(int(tempy*2/wally)-int(tempy/wally));
                tempy = tempy*tempy;
                tempz = a.getZ() - b.getZ();
                tempz = tempz - wallz*(int(tempz*2/wallz)-int(tempz/wallz));
                tempz = tempz*tempz;
            }

            temp = sqrt(tempx + tempy + tempz);

            return temp;
        }
```

```
        //return the distance between two atoms in the molecule
        double distance(int i, int j)
        {
            double dx, dy, dz;

            dx = atomContainer[i].getX() - atomContainer[j].getX();
            dy = atomContainer[i].getY() - atomContainer[j].getY();
            dz = atomContainer[i].getZ() - atomContainer[j].getZ();

            double d;

            d = sqrt(dx*dx + dy*dy + dz*dz);

            return d;
        }
        //return the intra energy of the molecule
        double intra(double kappa, double wallmin)
        {
            double energyIntra = 0;

            for(int i = 0; i < atomContainer.size()-1; i++)
            {
                for(int j = i+1; j< atomContainer.size(); j++)
                {
                    double d;
                    //calculate the distance between two atoms in the molecule
                    d = distance(i,j);
                    energyIntra +=
atomContainer[i].getCharge()*atomContainer[j].getCharge()/d -
atomContainer[i].getCharge()*atomContainer[j].getCharge()*realspace(d, kappa,
wallmin);
                }
            }

            return energyIntra;
        }

        //return the real space energy of the ewald potential
        double realspace(double r, double kappa, double wallmin)
        {
            double real;
```

```
        double a1 = 0.254829592;
        double a2 = -0.284496736;
        double a3 = 1.421413741;
        double a4 = -1.453152027;
        double a5 = 1.061405429;
        double pew = 0.3275911;
        double t, xsq, tp, rkap, erfc;
        rkap = (kappa/wallmin)*r;
        t = 1.0/(1.0 + pew * rkap);
        xsq = rkap * rkap;
        tp = t * ( a1 + t * ( a2 + t * ( a3 + t * ( a4 + t * a5 ) ) ) );
        erfc = tp*exp(-xsq);
        real = erfc/r;
        return real;
    }

    //show the atom information
    void showAtom(Atom m)
    {
        cout<<"Atom: "<<m.getName()<<endl;
        cout<<m.getX()<<endl<<m.getY()<<endl<<m.getZ()<<endl;

cout<<m.getCharge()<<endl<<m.getSig()<<endl<<m.getEps()<<endl;
    }

    //show the molecule information
    void showMolecule()
    {
        cout<<"Name: "<<name<<endl;
        cout<<"site: "<<siteNumber<<endl;
        cout<<"Mass: "<<mass<<endl;
        cout<<"Atom number: "<<atomContainer.size()<<endl;
        cout<<"Charge: "<<endl;
        for(int i = 0; i < chargeContainer.size(); i++)
            cout<<chargeContainer[i]<<" ";
        cout<<endl;
        cout<<"rx          ry          rz"<<endl;
        for(int i = 0; i < rx.size(); i++)
            cout<<rx[i]<<" "<<ry[i]<<" "<<rz[i]<<endl;
    }
```

```
//generate a random number [0, 1)
double ran()
{
    return double(rand())/(double(RAND_MAX) + double(1));
}

//clear all the container in the molecule
void clear()
{
    atomContainer.clear();
    rx.clear();
    ry.clear();
    rz.clear();
    chargeContainer.clear();
}
//return the energy between the molecule and the wall
double wallInteraction(double wallx, double wally, double wallz, double
twoPi, bool boundaryX, double thickx, bool boundaryY, double thicky, bool
boundaryZ, double thickz, double wallSigX, double wallEpsX, double wallSigY,
double wallEpsY, double wallSigZ, double wallEpsZ, double rhox, double rhoy,
double rhoz)
{
    double energyWall;
    energyWall = 0.0;

    double sigFS, epsFS;
    double factor;
    double tempWallEnergy;

    //calculate the energy for every atom in the molecule
    for(int i = 0; i < atomContainer.size(); i++)
    {
        if(atomContainer[i].getEps() == 0)
            continue;
        //calculate the energy from the wall if wall is at x direction
        if(!boundaryX)
        {
            sigFS = (atomContainer[i].getSig() + wallSigX)/2.0;
            epsFS = sqrt(atomContainer[i].getEps() * wallEpsX)/2.0;
            factor = twoPi*rhox*epsFS*sigFS*sigFS*thickx;
            double tempX;
```

```
                    tempX = wallx/2.0 - fabs(atomContainer[i].getX());
                    tempWallEnergy = factor*(0.4*pow(sigFS/tempX, 10.0) -
pow(sigFS/tempX, 4.0) - pow(sigFS, 4.0)/3/thickx/pow(0.61*thickx+tempX, 3.0));
                    tempX = fabs(wallx - tempX);
                    tempWallEnergy += factor*(0.4*pow(sigFS/tempX, 10.0) -
pow(sigFS/tempX, 4.0) - pow(sigFS, 4.0)/3/thickx/pow(0.61*thickx+tempX, 3.0));
                    energyWall += tempWallEnergy;
                }
                //calculate the energy from the wall if the wall is at y direction
                if(!boundaryY)
                {
                    sigFS = (atomContainer[i].getSig() + wallSigY)/2.0;
                    epsFS = sqrt(atomContainer[i].getEps() * wallEpsY);
                    factor = twoPi*rhoy*epsFS*sigFS*sigFS*thicky;
                    double tempY;
                    tempY = wally/2.0 - fabs(atomContainer[i].getY());
                    tempWallEnergy = factor*(0.4*pow(sigFS/tempY, 10.0) -
pow(sigFS/tempY, 4.0) - pow(sigFS, 4.0)/3/thicky/pow(0.61*thicky+tempY, 3.0));
                    tempY = fabs(wally - tempY);
                    tempWallEnergy += factor*(0.4*pow(sigFS/tempY, 10.0) -
pow(sigFS/tempY, 4.0) - pow(sigFS, 4.0)/3/thicky/pow(0.61*thicky+tempY, 3.0));
                    energyWall += tempWallEnergy;
                }
                //calculate the energy from the wall if the wall is at z direction
                if(!boundaryZ)
                {
                    sigFS = (atomContainer[i].getSig() + wallSigZ)/2.0;
                    epsFS = sqrt(atomContainer[i].getEps() * wallEpsZ);
                    factor = twoPi*rhoz*epsFS*sigFS*sigFS*thickz;
                    double tempZ;
                    tempZ = wallz/2.0 - fabs(atomContainer[i].getZ());
                    tempWallEnergy = factor*(0.4*pow(sigFS/tempZ, 10.0) -
pow(sigFS/tempZ, 4.0) - pow(sigFS, 4.0)/3/thickz/pow(0.61*thickz+tempZ, 3.0));
                    tempZ = fabs(wallz - tempZ);
                    tempWallEnergy += factor*(0.4*pow(sigFS/tempZ, 10.0) -
pow(sigFS/tempZ, 4.0) - pow(sigFS, 4.0)/3/thickz/pow(0.61*thickz+tempZ, 3.0));
                    energyWall += tempWallEnergy;
                }
            }

            return energyWall;
        }
```

```
        //Accessor
        //get the atom number in the molecule
        int getSiteNumber() {return siteNumber;}

        //get the name
        string getName() {return name;}

        //get the mass
        double getMass() {return mass;}

        //get the fixing
        bool getFixing() {return fixing;}

        //get the atom container
        vector<Atom> getContainer() {return atomContainer;}

        //get the charge container
        vector<double> getChargeContainer() {return chargeContainer;}

        //get the rx container
        vector<double> getRX() {return rx;}

        //get the ry container
        vector<double> getRY() {return ry;}

        //get the rz container
        vector<double> getRZ() {return rz;}

        //get the atom
        Atom getAtom(int i) {return atomContainer[i];}

        //get the distance between two molecules
        double centralDistance(const Molecule &m, double rangeX, double
rangeY, double rangeZ)
        {
            Atom temp1, temp2;
            //take the position of the first atom in the moleucle as the position of
the molecule
            temp1 = atomContainer[0];
            temp2 = m.atomContainer[0];
```

```
            double dist, distx, disty, distz;
            distx = temp1.getX()-temp2.getX();
            distx = distx-rangeX*(int(distx*2/rangeX)-int(distx/rangeX));
            distx = distx*distx;
            disty = temp1.getY()-temp2.getY();
            disty = disty-rangeY*(int(disty*2/rangeY)-int(disty/rangeY));
            disty = disty*disty;
            distz = temp1.getZ()-temp2.getZ();
            distz = distz-rangeZ*(int(distz*2/rangeZ)-int(distz/rangeZ));
            distz = distz*distz;
            dist = sqrt(distx+disty+distz);
            return dist;
        }
        //output the coordinates to the file by pdb format
        void moleculePdb(ofstream &file, int num)
        {
            for(int i = 0; i < atomContainer.size(); i++)
            {
                atomContainer[i].atomPdb(file, num*siteNumber+i);
            }
        }
};
#endif

/*Atom.h*/
/*Descript the property of the atom*/
#ifndef ATOM_H
#define ATOM_H
#include <iostream>
#include <string>
#include <iomanip>
using namespace std;

class Atom
{
    private: double x, y, z;//represent the coordinates of the atom
        string name;//atom name
        double charge, sig, eps;//charge, sigma and epsilon in Lennard-Johns
potential
    public:
        //Constructor
```

```
        Atom()
        {
            x = 0.0; y = 0.0; z = 0.0;//initialize the atom to the center of the
system
            name = "none";//set the name of the atom to be "none"
            charge = 0.0;//set the charge to be zero
            sig = 0;//set the sigma to be zero
            eps = 0;//set the epsilon to be zero
        }

        //input the name, charge, sigma and epsilon to the atom
        Atom(string n, double c, double s, double e)
        {
            name = n;
            charge = c;
            sig = s;
            eps = e;
        }

        //set the name, coordinates, charge, sigma and epsilon
        Atom(string n, double cx, double cy, double cz, double c, double s,
double e)
        {
            x = cx; y = cy; z = cz;
            name = n;
            charge = c;
            sig = s;
            eps = e;
        }

        //Mutator
        //save the name, coordinates, charge, sigma and epsilon of the atom to the
file
        void save(ofstream &file)
        {
            //set 5 width for name and 10 significant figures for the value of the
parameter
            file<<setw(5)<<name<<" "<<setprecision(10)<<x<<" "<<y<<"
"<<z<<" "<<charge<<" "<<sig<<" "<<eps<<endl;
        }
```

```
//read the name, coordinates, charge, sigma and epsilong from the file
void read(ifstream &file)
{
	file>>name>>x>>y>>z>>charge>>sig>>eps;
}

//set the x value
void setX(double cx)
{
	x = cx;
}

//set the y value
void setY(double cy)
{
	y = cy;
}

//set the z value
void setZ(double cz)
{
	z = cz;
}

//set the coordinates
void setCoordination(double cx, double cy, double cz)
{
	x = cx;
	y = cy;
	z = cz;
}

//set the name
void setName(string n)
{
	name = n;
}

//set the charge
void setCharge(double c)
{
```

```
	charge = c;
}

//set the sigma
void setSig(double s)
{
	sig = s;
}

//set the epsilon
void setEps(double e)
{
	eps = e;
}

//shift the atom
void shift(double dx, double dy, double dz)
{
	x += dx;
	y += dy;
	z += dz;
}

//define the minus between atoms
Atom operator-(Atom &right)
{
	Atom temp(name,charge,sig,eps);
	temp.x = x - right.x;
	temp.y = y - right.y;
	temp.z = z - right.z;

	return temp;
}

//define the plus between atoms
Atom operator+(Atom &right)
{
	Atom temp(name,charge,sig,eps);
	temp.x = x + right.x;
	temp.y = y + right.y;
	temp.z = z + right.z;
```

```
        return temp;
    }

    //define the equal, copy the properties of one atom to another one
    const Atom operator=(const Atom &right)
    {
        name = right.name;
        x = right.x;
        y = right.y;
        z = right.z;
        charge = right.charge;
        sig = right.sig;
        eps = right.eps;
        return *this;
    }


    //rotate the atom by a matrix
    Atom rotate(double **rotate, int row)
    {
        Atom temp(name,charge,sig,eps);

        temp.x = x*rotate[0][0] + y*rotate[1][0] + z*rotate[2][0];
        temp.y = x*rotate[0][1] + y*rotate[1][1] + z*rotate[2][1];
        temp.z = x*rotate[0][2] + y*rotate[1][2] + z*rotate[2][2];

        return temp;
    }

    //move the atom into the system if the atom is out of system when do
movement trial
    void align(double bx, int isX, double by, int isY, double bz, int isZ)
    {
        if(isX == 1)
            x -= bx;//bx is the length of box at x direction
        else if(isX == -1)
            x += bx;
        if(isY == 1)
            y -= by;//by is the length of box at y direction
        else if(isY == -1)
            y += by;
```

```
            if(isZ == 1)
                z -= bz;//bz is the length of box at z direction
            else if(isZ == -1)
                z += bz;
        }

        //show atom information
        void showAtom()
        {
            cout<<name<<" "<<x<<" "<<y<<" "<<z<<" "<<charge<<"
"<<sig<<" "<<eps<<endl;
        }

        //input the atom coordinates into the file by pdb formate
        void atomPdb(ofstream &file, int num)
        {
            file<<"HETATH"<<setw(5)<<num+1<<"
"<<setw(3)<<left<<name<<"
"<<setw(8)<<setprecision(3)<<fixed<<right<<x<<setw(8)<<y<<setw(8)<<z<<en
dl;
        }

        //Accessor
        //get the x value
        double getX() {return x;}

        //get the y value
        double getY() {return y;}

        //get the z value
        double getZ() {return z;}

        //get the name of the atom
        string getName() {return name;}

        //get the charge of the atom
        double getCharge() {return charge;}

        //get the sigma of the atom
        double getSig() {return sig;}
```

```
        //get the epsilon of the atom
        double getEps() {return eps;}

        //Deconstructor
        ~Atom()
        {
        }
};
#endif
```

APPENDIX B

*Input file, model file and output file in above code
*Molecule name atom number mass fix
*first atom name x y z charge σ ε
*second atom name x y z charge σ ε
*……

water	3	18.016	0			
O	-0.2121	7.973608	0.740273	-15.4476	3.1655	0.1554
H	-1.20721	7.887144	0.78791	7.7238	0	0
H	0.082794	7.909364	-0.2131	7.7238	0	0
water	3	18.016	0			
O	-3.59385	3.193661	1.719466	-15.4476	3.1655	0.1554
H	-4.43107	3.323236	1.188178	7.7238	0	0
H	-3.62128	3.775873	2.532044	7.7238	0	0
water	3	18.016	0			
O	-2.79486	5.114584	-2.8305	-15.4476	3.1655	0.1554
H	-2.47879	4.581979	-2.04538	7.7238	0	0
H	-2.90301	6.071816	-2.56214	7.7238	0	0
water	3	18.016	0			
O	0.286635	8.163975	-1.92878	-15.4476	3.1655	0.1554
H	0.542527	9.024783	-2.3687	7.7238	0	0
H	-0.62913	7.896039	-2.22808	7.7238	0	0
water	3	18.016	0			
O	-2.88028	7.372825	-1.87054	-15.4476	3.1655	0.1554
H	-3.44055	8.078696	-2.30396	7.7238	0	0
H	-2.97155	7.438315	-0.87687	7.7238	0	0
water	3	18.016	0			
O	-5.72574	4.302854	0.304639	-15.4476	3.1655	0.1554
H	-5.79993	5.195709	-0.13955	7.7238	0	0
H	-6.22554	3.621792	-0.23049	7.7238	0	0
water	3	18.016	0			
O	0.25118	-10.8405	2.851454	-15.4476	3.1655	0.1554
H	1.0511	-10.9713	3.437118	7.7238	0	0
H	0.148331	-9.86849	2.639986	7.7238	0	0
water	3	18.016	0			
O	-1.02647	10.48524	-3.10571	-15.4476	3.1655	0.1554
H	-1.6923	10.94915	-2.52136	7.7238	0	0
H	-0.10202	10.75811	-2.83937	7.7238	0	0

……

APPENDIX C

```
c Generate the nanotube
c The nanotube is a 'armchair' type, here take 6*6 and all the six in a
c same plane
c cc is the length between two neighbor carbon
c n_cyc is the number of cychexane of every around
c n_axis is the number of cychexane along the axis of the nanotube
c radii of the nanobute cc*sqrt(3)/(2*sin(180/n))
c The axis of the nanotube is the z
      program generate
      implicit none
      include 'parameter'
      integer i
      double precision x,y,z
      double precision z_b,z_t,r_in
      double precision length,radii
      open(6,file='coordiante',status='unknown')
      radii=cc*sqrt(3.0)/(2*sin(real(3.1415926/n_cyc)))
      x=0.0
      y=0.0
      z_t=cc/2
      z_b=0
      write(6,*) '     x      ','     y      ','   z        '
      call bottom(x,y,z_b)
      call top(x,y,z_t)
      do 13 i=1,n_axis
      if (mod(i,2) .eq. 1) then
      z_t=z_t+cc*1.5
      call top(x,y,z_t)
      z_b=z_b+cc*1.5
      call bottom(x,y,z_b)
      end if
      if (mod(i,2) .eq. 0) then
      z_b=z_b+cc*1.5
      call bottom(x,y,z_b)
      z_t=z_t+cc*1.5
      call top(x,y,z_t)
      end if
   13 continue
      open(7,file='characters',status='unknown')
      length=cc/2+cc*1.5*n_axis
      r_in=radii*cos(3.1415926/n_cyc)
```

```
      write(7,*) 'length of the nanotube',length+2*C_radii
      write(7,*) 'diameter',2*(radii-C_radii)
      write(7,*) 'C-C bond length',cc
      write(7,*) 'total number of carbon',(1+n_axis)*2*n_cyc
      end program generate
cccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine bottom(x,y,z)
      include 'parameter'
      integer i
      double precision radii
      double precision x,y,z
      radii=cc*sqrt(3.0)/(2*sin(real(3.1415926/n_cyc)))
      do 11 i=1,n_cyc
      x=radii*cos(real(2*3.1415926*(i-1)/n_cyc))
      y=radii*sin(real(2*3.1415926*(i-1)/n_cyc))
      if (abs(x) .LT. 0.001) x=0.00000000000
      if (abs(y) .LT. 0.001) y=0.00000000000
      write(6,91) x,y,z
   11 continue
   91 format (F8.3,F8.3, F8.3)
c
      end
ccccccccccccccccccccccccccccccccccccccccccccccccccccccccc
ccccccccccccccccccccccccccccccccccccccccccccccccc
      subroutine top(x,y,z)
      include 'parameter'
      integer i
      double precision radii
      double precision x,y,z
      radii=cc*sqrt(3.0)/(2*sin(real(3.1415926/n_cyc)))
      do 12 i=1,n_cyc
      x=radii*cos(real(3.1415926*(2*i-1)/n_cyc))
      y=radii*sin(real(3.1415926*(2*i-1)/n_cyc))
      if (abs(x) .LT. 0.001) x=0.000000000000
      if (abs(y) .LT. 0.001) y=0.000000000000
      write(6,92) x,y,z
   12 continue
c
   92 format(F8.3,F8.3, F8.3)
c
      end
```