Fall 2008

A coarse-grain molecular dynamics study of the nanotribological properties of nanoparticle solutions

Ramesh Chembeti

Department:

A COARSE-GRAIN MOLECULAR DYNAMICS STUDY

OF

THE NANOTRIBOLOGICAL PROPERTIES OF NANOPARTICLE SOLUTIONS

by

RAMESH CHEMBETI

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN CHEMICAL ENGINEERING

2008

Approved by

Dr. Jee-Ching Wang, Advisor
Dr. Parthasakha Neogi
Dr. Louis Ge

ABSTRACT

In this study, solutions of alkanethiol-capped nanoparticles in alkane are examined using molecular dynamics simulations for their nanotribological potential based on the hypothesis that fluid molecules of very different sizes may interrupt each other's layering tendency to result in less layered or non-layered configurations and provide better lubrication for nanodevices. An effective nanoparticle-nanoparticle pair potential based on previous atomistic approach is used and the temperature and parallel pressure are controlled in place of chemical potential for defining thermodynamic state. When compressed, the confined nanoparticle-containing alkane films generate reduced oscillations in perpendicular forces and smoother expansion in lateral dimensions, indicating lesser extent of layering due to the presence of much bigger nanoparticles. The nanoparticles are found to be well dispersed by the alkane solvent throughout all separations, meaning no or little tendency to form clusters or aggregate towards the confining surfaces, which is important for the stability and quality of the nanoparticle solutions as nanotribological lubricant. When sheared by a sliding surface, the confined fluids tend to move in the same parallel direction so that their density profiles remain practically unchanged. The shear stress resulting from the sliding surface has been calculated and found to increase with faster sliding speed but not proportionally. More importantly, the presence of the nanoparticles in the lubricant films reduces the shear stress noticeably and thereby reducing the apparent viscosity and frictional force. This effect is particularly evident under relatively large sliding speed and large surface separations. Regarding mobility, the nanoparticles exhibit lower diffusivity in nanoconfinement than typical fluids and their diffusivity can be enhanced by shearing.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Dr. Jee-Ching Wang for being very patient and helpful throughout the course of my thesis. I would like to thank Dr. Parthasakha Neogi and Dr. Louis Ge for spending their valuable time as committee members. Financial support from Missouri Research Board and National Science Foundation is greatly acknowledged. My heartfelt thanks to my parents, my sister and all my friends for being there for me all the time.

.

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

1. INTRODUCTION

1.1 NANOTRIBOLOGY

A nanometer (nm) is 10^{-9} of a meter and equivalent to 10 angstrom (Å). It is roughly 4-5 times the size of a typical atom. Recently a scientific and technological revolution has begun to systematically study, manipulate, and devise matter on the nanometer length scale. The terms "nanoscience" and "nanotechnology" have generally been used to represent such efforts. As nicely explained by Eric Drexler in his book "Engines of Creation" [1] and by Richard Feynman in his lecture "There's plenty of room at the bottom" [2], atoms are the root cause for everything and the ability to deal with individual atoms and molecules is the basis for nanotechnology. While nanoscale devices based on moving atomic/molecular components have the potential to drastically alter and improve technologies for energy transfer, data storage, drug delivery, computing, chemical manufacture, and so on, they can unfortunately be very vulnerable to friction and wear due to their extremely small sizes. From a fundamental point of view, friction originates from atomic interactions/forces between moving parts, which resist the motion and jiggle the atoms to generate heat and cause structural deformation (wear). While friction and wear may be as simple an issue as added material and energy costs to conventional macroscale devices and structures, they could turn envisioned nanotechnologies into unrealizable dreams or substantially reduce the working life of nanodevices and nanostructures because the resultant stress per unit volume can be too much to bear. Nanoelectromechanical systems (NEMS) and microelectromechanical systems (MEMS) are good examples.

The term “tribology” is derived from the Greek words “tribo” meaning rubbing and “logy” meaning knowledge. Its original application by the Greeks was to understand the motion of large stones sliding across the earth's surface. Today tribology has grown into a field that deals with all issues involving friction, wear, and lubrication. Lubricants are substances interposed between two surfaces in relative motion for the purpose of reducing the friction and/or the wear between them. In decreasing order of lubricant film thickness, lubrication has traditionally been divided into three regimes [3]: *hydrodynamic* or *bulk*, *mixed* or *intermediate*, and *boundary*. The past few decades have seen increasing miniaturization of device components, the advent of modern surface proximity probes, and extension of boundary lubrication into nanometer scales. All these developments are pertinent to the chemical and mechanical stability of the moving nanostructures and have inspired the birth of a new field, namely nanotribology [4] or molecular tribology.

1.2 NANOCONFINED FLUIDS

It is easy to see that in nanotribology the lubricant films are confined to highly restricted geometries whose dimensions are nanoscale and comparable to a few molecular diameters. It should not be difficult to perceive that such molecularly thin films could have drastically different behavior than the same materials in the unconstrained bulk phase. Recent experimental studies using surface force apparatus (SFA) [5-9], atomic force microscopy (AFM) [10-12], friction force microscopy (FFM) [13,14], and quartz crystal microbalance (QCM) [15-17] have confirmed the significant effects of the confining surfaces on the properties of nanoscopically confined fluid films and further suggested that nanoscale confinement can induce fluid molecules to form layered

configurations, causing nanoconfined fluids to have very different properties than those of bulk fluids, where molecules possess no preferred orientation. Explicit evidences at the molecular level have come from computer molecular simulation studies using Monte Carlo (MC) [18-23] and molecular dynamics (MD) [24-30] techniques. Other theoretical approaches such as density functional theory [31,32] and integral equation theory [33,34] have also been extended to analyze nanoconfined fluids and confirm the formation of fluid layers under confinement.

It has now been well established that when the separation/spacing between the confining surfaces is larger than about 10 molecular diameters, confined fluids behave much the same as bulk fluids in many aspects. As the separation decreases, molecular orientation and fluid configuration undergo changes and, more interestingly and importantly, the isotropy between the perpendicular and parallel directions breaks down. From thermodynamic point of view, such configurational changes are entropy changes, which, according to $(\partial S/\partial V)_{U,N} = P/T$, cause the pressure of a nanoconfined fluid to not only change but also become different in the perpendicular and parallel directions. The difference between the perpendicular pressure ($P_{\perp}$) and parallel pressure ($P_{\parallel}$) has been measured as *solvation force* (f_s) by the surface force apparatus (SFA) [5-9], where the confined fluid is open to the bulk reservoir under isothermal-isobaric condition and the parallel pressure as a result is taken to be the same as the bulk pressure. Interestingly, when symmetric molecules are confined under a constant bulk/parallel pressure, solvation/surface force oscillates as a function of surface separation with a periodicity that is equivalent to the mean molecular diameter [cf. Fig. 1(b)]. This behavior is not in accord with the expectations from conventional continuum theories such as lubrication

theory [35,36] and DLVO theory [37,38], but signifies the importance of the discrete excluded volume of the fluid molecules at small separations. Specifically, nanoscale confinement and exclude volume can work together to pack symmetric fluid molecules into layers parallel to the confining surfaces and one oscillation in the surface/solvation force corresponds to an increase or decrease of one molecular layer in the confined fluid film. As the surface separation becomes smaller, the *layering* phenomenon and the force oscillation become stronger.

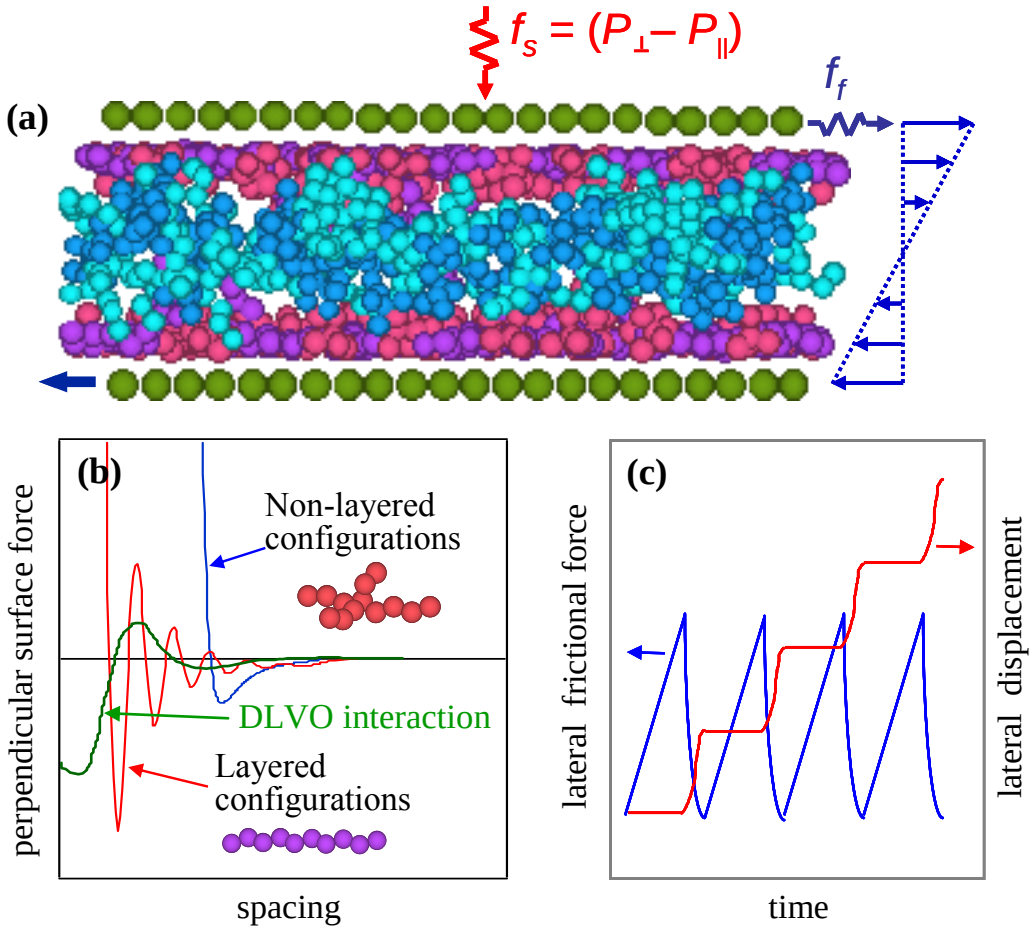

Figure 1.1. Schematics of (a) a nanoconfined fluid, surface force, f_s, and frictional force, f_f, (b) solvation/surface force profiles for confined fluids with layered or non-layered configurations, and (c) stick-slip motion of confined fluids with layered configurations.

In relation to nanotribology, when a strongly layered confined fluid is sheared, instead of lubricated smooth motion, it exhibits yield stress, stick-slip motion [cf. Fig. 1(c)], and apparent shear viscosity that could be orders of magnitude higher than bulk values [39,40]. It should be emphasized that the stick-slip motion is not caused by surface asperities as would be explained by the traditional "rough surface" model. These solid-like responses reflect the solid-like nature of the layered configurations, which could be linked to the peaked, repulsive surface forces [cp. Fig. 1(a)]. It is important to emphasize here that the layered configurations of nanoconfined lubricant films can cause undesirably large friction and wear and consequently significant material and energy costs. On the other hand, between two peaked surface forces and during the slip phase, nanoconfined fluids undergo configurational changes to become less layered, thereby exhibiting lubricated smooth motion and reduced friction and wear.

1.3 SOLUTIONS OF ALKANETHIOL-CAPPED NANOPARTICLES IN ALKANE

The above observations and discussion suggest that more desirable nanotribological properties can be obtained from lubricants that would be more resistant to being layered in nanoconfinement. One source of such resistance can come from structural asymmetry. Indeed, branched alkanes have been shown to exhibit reduced or even no oscillation in their solvation/surface forces measured by SFA [6,23], which is understood to be because of the side branches that disrupt the formation of layered configurations. More generally, it can be reasoned that the proper lubricants for nanotribology should be those that have strong intrinsic means to resist the layering tendency under nanoconfinement. This research considers one such possibility, namely the hypothesis that molecules of

sufficiently different sizes, when mixed together, could disrupt each other's layering tendency in nanoconfined space to result in less layered or even non-layered configurations to satisfy nanotribological needs. One necessary condition worth mentioning is that there should be good solubility between the molecules in order to not have phase separation in nanoconfinement. In this research work, we propose to examine the solution of alkanethiol-capped nanoparticles in alkane.

Liquid alkanes (C_nH_{2n+2}) are oils and very commonly used lubricants. They were also one of the first nanoconfined systems to be studied by SFA, but the results suggested that liquid alkanes alone may not be good lubricants for moving nanodevices and nanostructures. Here we hypothesize that their nanotribological properties could be improved by adding alkanethiol ($C_nH_{2n+1}SH$)-capped nanoparticles whose sizes are several times larger than alkane molecules. These nanoparticles represent a recent breakthrough and have been touted as one the most important ingredients for nanotechnologies [41-43]. As depicted in Figure 2, they are charge-neutral composites with an inorganic crystalline core capped in a dense shell of alkanethiol molecules. The capping molecules have been called *surfactants* because they have a polar head group (e.g. S) that strongly bonds to the core and nonpolar alkyl chains that disperse the nanoparticles in nonpolar alkane solutions. Today, it has become virtually a routine to synthesize such nanoparticles. As mentioned earlier, the good solubility and the significant size mismatch could impede the formation of layered configurations, thereby making the solutions of alkanethiol-capped nanoparticles in alkane promising viable lubricants for nano- and micro-scale systems.

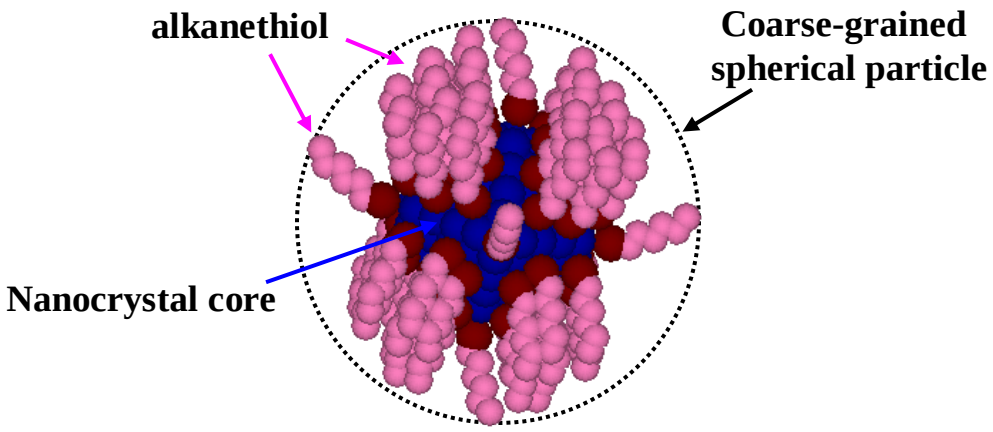

Figure 1.2. Ball model and a coarse-grained representation of an alkanethiol-capped nanoparticle.

1.4 PROPOSED RESEARCH METHODOLOGY

Despite fast progress, modern experimental techniques still have limited resolutions and difficulties in accessing the physics of confined complex fluids at the nanoscale to evaluate their nanotribological potential. Theoretical studies of nanoconfined fluids using density functional theory [31,32], integral equation theory [33,34], Enskog theory [44], and functional perturbation theory [45] have advanced our understanding, but their current status is still limited to simple atomic or highly idealized fluids. For complex nanoconfined fluid systems, computer molecular simulation has become a major and sometimes preferred research method [13-30]. In this project, MD simulation technique [46, 47] is adopted for its better handling of dynamic properties and complex simulation models. A number of important variables are investigated including nanoparticle loading, surface separation, and shear rate. For computational efficiency and as a first

attempt, coarse grained simulation models are considered. Details of the employed methodology are provided in the next section.

It is worth noting that the future of scientific computing and molecular simulation has shifted from expensive supercomputers to PC-based systems for their superior performance/cost ratio, much increased speed and stability, and excellent expandability. This trend is moving fast in both academia and industries. As an added value, this project runs the same simulation codes on both PC and more conventional IBM workstation in order to make comparison and pave the way for future work.

.

2. SIMULATION METHODOLOGY

2.1 MOLECULAR DYNAMICS COMPUTER SIMULATION TECHNIQUE

Molecular dynamics (MD) simulation is a technique for computing equilibrium and transport properties for classical many-body systems. The word classical means that the motion of the constituent particles (e.g. atoms or pseudo-atoms) obeys the laws of classical mechanics (e.g. $\dot{\mathbf{r}}_i = \mathbf{v}_i$, $\ddot{\mathbf{r}}_i = \mathbf{a}_i$). This is a reasonable and often excellent approximation for a wide range of systems and properties. In this context, every phenomenon and every property can be traced back to the coordinates and momenta of the constituent particles. The essence of the MD method is to numerically integrate the equation of motion of classical mechanics for every particle in the model system and the direct simulation results are the coordinates and momenta of all moving particles at different time instants. This is equivalent to generating many microstates in the same ensemble (controlled macroscopic conditions) and permits the use of statistical mechanically derived equations to complete property calculation. In general, MD can be applied to identify molecular origins, test hypotheses, estimate missing or unreliable data, and characterize the relative importance of different parameters and variables. MD thus bridges between and complements both theoretical and experimental approaches. It is of particular value for systems that are too complicated to be studied by first principles or too difficult to experimental studies.

In general, the equation of motion of classical mechanics for a specific particle *i* can be expressed as a second-order differential equation,

$$m_i \ddot{\mathbf{r}}_i = \mathbf{f}_i + \mathbf{g}_i \ , \tag{2.1}$$

where m_i and $\ddot{\mathbf{r}}_i$ are the mass and acceleration of the particle i. $\boldsymbol{f_i}$ and $\boldsymbol{g_i}$ denote the total force on particle i from other particles and the extra force(s) due to external constraint(s) imposed on the system. Equivalently, Eq. (2.1) can be converted into two first-order differential equations,

$$\begin{cases} m_i\dot{\mathbf{r}}_i = \mathbf{p}_i + \mathbf{g}_i^r & \mathbf{r}_i \\ \dot{\mathbf{p}}_i = \mathbf{f}_i + \mathbf{g}_i^p & \mathbf{p}_i \end{cases} \Rightarrow \begin{cases} \dot{\mathbf{r}}_i = \mathbf{v}_i + \dfrac{1}{m_i}\mathbf{g}_i^r & \mathbf{r}_i \quad (2.2) \\ \dot{\mathbf{v}}_i = \mathbf{a}_i + \dfrac{1}{m_i}\mathbf{g}_i^p & \mathbf{p}_i \quad (2.3) \end{cases}$$

Here $\mathbf{g}_i^r$ and $\mathbf{g}_i^p$ are the corresponding constraint forces acting on the coordinate ($\mathbf{r}_i$) and momentum ($\dot{\mathbf{r}}_i$). $\boldsymbol{f_i}$ is derived from the total potential energy $U(\mathbf{r})$ of the system,

$$\mathbf{f}_i = -\nabla_{\mathbf{r}_i} U(\mathbf{r}), \quad (2.4)$$

where $U(\mathbf{r})$ is generally constructed by pair-wise additions of two types of contributions,

$$U(\mathbf{r}) = U^{inter}(\mathbf{r}) + U^{intra}(\mathbf{r}), \quad (2.5)$$

where $U^{inter}(\mathbf{r})$ is the sum over all interactions between atoms in different molecules and $U^{intra}(\mathbf{r})$ is the sum over interactions between atoms within the same molecule. The nanoconfined fluids considered in this project are consisted of Lennard-Jones (LJ) particles whose $U^{intra}(\mathbf{r}) = 0$ and $U^{extra}(\mathbf{r})$ will be discussed in detail below.

2.2 SIMULATION APPROACH AND MODELS

In this project, we consider hexanethiol-capped gold nanoparticle (cf. Figure 1.2), $Au_{140}(SC_6H_{13})_{62}$, to be the model nanoparticle. This nanoparticle has recently been

simulated with atomistic models in our group [48] and its size and interaction characteristics have been studied and formulated into a Lennard-Jones 12-6 potential

$$U_{\text{L-J}} = 4\varepsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6}\right] , \tag{2.6}$$

For computational consistency and efficiency, we also opted for an alkane as the solvent that can be approximated as a spherical particle and has been modeled as a LJ particle. Cyclohexane is chosen for this purpose and its LJ parameters are adopted from the literature [49] and tabulated in Table 2.1 together with those for nanoparticles.

The main role of the confining surfaces is to create nanoscale confinement for lubricant films. In principle, as long as the fluid molecules can be layered, the layering phenomenon, surface force oscillations, and other relevant issues persist regardless of the confining surfaces being structured or structure less [50-52] and having attractive or repulsive interaction [52,53-55] with the confined fluid. Since a relatively large portion of the envisioned nanotechnologies involve silicon (Si), this project considers the Si(111) surface as the confining surfaces. Silicon has the diamond structure with a lattice constant equal to 5.43 Å. Figure 2.1 shows the ball models of different views of silicon.

Table 2.1. Lennard-Jones 12-6 potential parameters.

Particles	ε (J/mol)	σ (Å)
$Au_{140}(SC_6H_{13})_{62}$	10827.08×R	29.000
Cyclohexane	484.00×R	5.466
Si	202.45×R	3.826

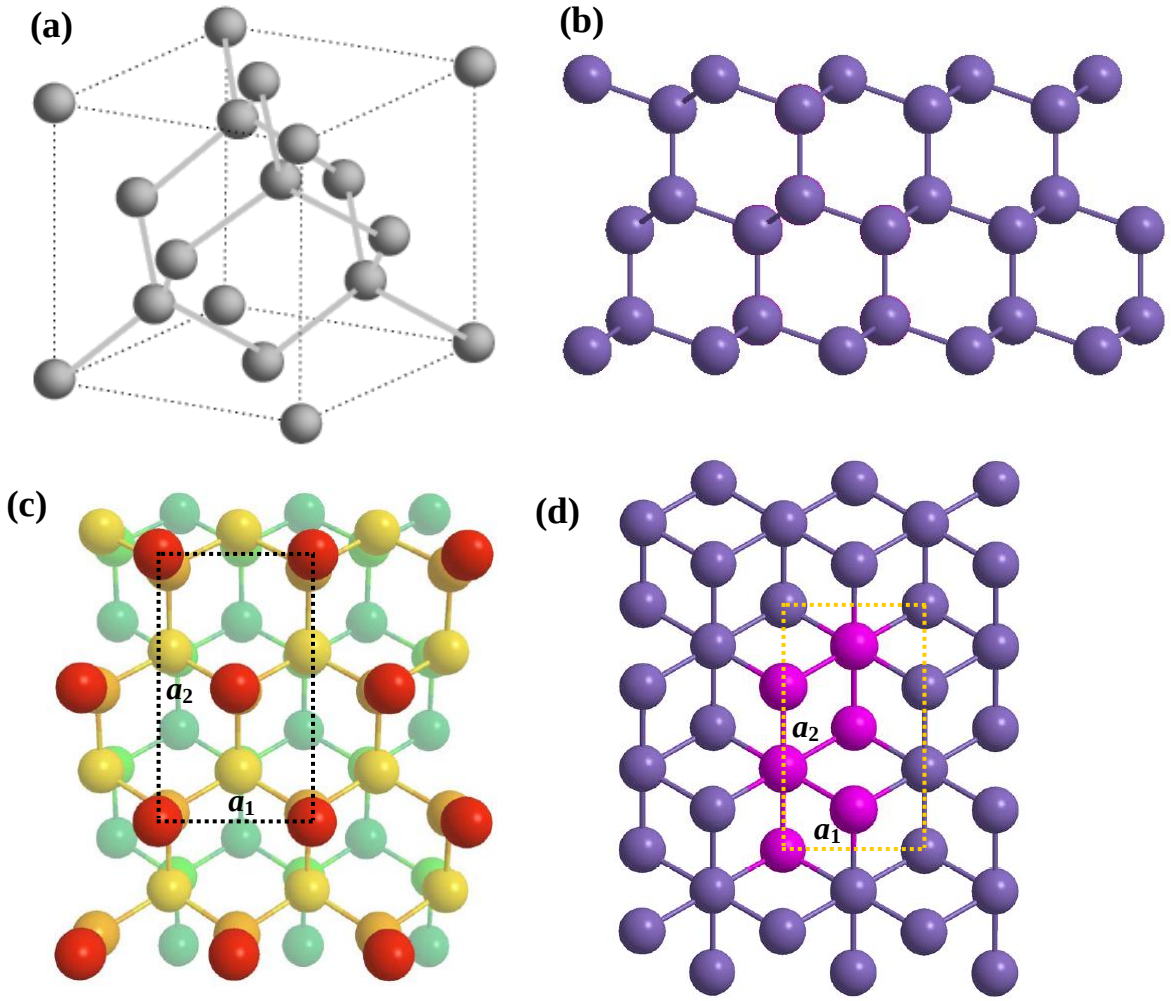

Figure 2.1. Ball models of (a) silicon diamond structure, (b) a side view of the Si(111) surface, (c) a top-down view of the Si(111) surface with depth fading indicated by different colors, and (d) a top-down view of the Si(111) surface with pink balls and the rectangle showing one surface unit cell. a_1 and a_2 are the dimensions of a surface unit cell and equal to 3.84 Å and 6.65 Å, respectively.

To systematically simulate nanoconfined nanoparticle-alkane solutions under different conditions while allowing meaningful comparison, the thermodynamic state of the confined fluids should be defined and controlled in our simulation studies. Since the confined fluids in SFA are open to and in equilibrium with the bulk reservoir, temperature (T) and chemical potential (μ) are thus the most natural thermodynamic

variables for this purpose. However, evaluating and controlling chemical potential is extremely difficult for MD simulations, especially when complex systems and conditions are considered. Alternative approaches to overcome this difficulty have been developed recently and termed the *NAPT* [56,57] and *NhPT* methods [30]. The essence of these two methods is to constrain the temperature (T) and the pressure (or stress) parallel to the confining surfaces ($P_{\parallel}$) to prespecified values to offer a virtual isothermal-isobaric reservoir for confined fluids to reach equilibrium with. In classical statistical mechanics and molecular simulation, T and P can be expressed as follows as function of particle coordinates and momenta,

$$\left\langle \frac{1}{2m} p^2 \right\rangle = \left\langle \frac{1}{2} m v^2 \right\rangle = \frac{3}{2} k_{\mathrm{B}} T \Rightarrow T = \sum_{i=1}^{N} \frac{m_i \mathbf{v}_i^2}{3N k_{\mathrm{B}}} \quad , \tag{2.7}$$

$$P_{\parallel} = \frac{1}{2\ Ah} \sum_{\lambda = x,\, y} \left[\sum_i m_i \ \ \mathbf{v}_i^{\lambda} \cdot \mathbf{v}_i^{\lambda} \ + \sum_i \sum_{j>i} \mathbf{f}_{ij}^{\lambda} \cdot \mathbf{r}_{ij}^{\lambda} + \sum_i \sum_s \mathbf{f}_{is}^{\lambda} \cdot \mathbf{r}_{is}^{\lambda} \right] \ , \tag{2.8}$$

$$P_{\perp} = \frac{1}{Ah} \left[\sum_i m_i \ \ \mathbf{v}_i^{z} \cdot \mathbf{v}_i^{z} \ + \sum_i \sum_{j>i} \mathbf{f}_{ij}^{z} \cdot \mathbf{r}_{ij}^{z} + \sum_i \sum_s \mathbf{f}_{is}^{z} \cdot \left(\mathbf{r}_i^{z} \pm \frac{h}{2} \right) \right] \ , \tag{2.9}$$

where $\mathbf{v}_i$ is the velocity of a fluid particle and $\mathbf{f}_{ij}$, $\mathbf{f}_{is}$ are the interaction forces between a pair of particles separated by $\mathbf{r}_{ij} = \mathbf{r}_i - \mathbf{r}_j$ and $\mathbf{r}_{is} = \mathbf{r}_i - \mathbf{r}_s$. Here $\mathbf{r}_s$ represents the coordinate of a Si atom and $-h/2$ is for the upper confining substrate located at $z = h/2$ and $+h/2$ is for the lower one at $z = -h/2$. Both methods are 2D extensions of the original 3D *NPT* simulation method and constrain the total number of particles (N) and temperature (T). However, $P_{\parallel}$ is controlled in the *NhPT* method [30] by adjusting the lateral dimensions (i.e. surface area A) of the confined space and the lateral (x,y) coordinates of the fluid

particles, and in the *NAPT* method [56,57] by adjusting the separation (h) and the perpendicular (z) coordinates of the fluid particles. It should be noted that these two methods are equivalent to each other under the same conditions when time/ensemble averaged properties are compared. In the project, the *NhPT* method is employed because we are interested in studying the responses of confined nanoparticle-alkane solutions to continuous compression at constant speeds.

Since the parallel pressure constraint is imposed in the x and y directions, the equations of motion in the two lateral directions take on the *NPT* form [30,46,56,57],

$$\dot{\mathbf{r}}_i^{\lambda} = \mathbf{p}_i^{\lambda}/m_i + \chi\ \mathbf{r},\mathbf{p}\ \mathbf{r}_i^{\lambda}\ , \tag{2.10}$$

$$\dot{\mathbf{p}}_i^{\lambda} = \mathbf{f}_i^{\lambda} - \chi\ \mathbf{r},\mathbf{p}\ \mathbf{p}_i^{\lambda} - \xi_{\|}\ \mathbf{r},\mathbf{p}\ \mathbf{p}_i^{\lambda}\ , \tag{2.11}$$

$$\dot{S}_{\lambda} = \chi\ \mathbf{r},\mathbf{p}\ S_{\lambda}\ , \tag{2.12}$$

where S_{λ} denotes the lateral dimensions of a simulation box and $\chi\ \mathbf{r},\mathbf{p}$ is a dilation coefficient evaluated instantaneously to adjust S_{λ} or equivalently surface area A to achieve constant $P_{\|}$. In the perpendicular z direction, the equations of motion assume the *NVT* form,

$$\dot{\mathbf{r}}_i^{z} = \mathbf{p}_i^{z}/m_i\ , \tag{2.13}$$

$$\dot{\mathbf{p}}_i^{z} = \mathbf{f}_i^{z} - \xi_{\perp}\ \mathbf{r},\mathbf{p}\ \mathbf{p}_i^{z}\ . \tag{2.14}$$

$\xi_{\|}\ \mathbf{r},\mathbf{p}$ and $\xi_{\perp}\ \mathbf{r},\mathbf{p}$ are friction coefficients for temperature control and evaluated instantaneously using the Nose-Hoover method [46,58]. Proper evaluation of χ is more

difficult and has been accomplished via the loose coupling method of Berendsen *et al.* [46,59],

$$\chi = -\frac{\kappa_{\|}\ P_{\|,\mathrm{set}} - P_{\|}}{2t_P}\quad , \tag{2.15}$$

$$\frac{1}{\kappa_{\|}} = P_{\|} - \frac{1}{4Ah}\left\{\sum_i\sum_{j>i}\left[\frac{X(r_{ij})}{r_{ij}^4}\ \mathbf{r}_{ij}^{\lambda}\cdot\mathbf{r}_{ij}^{\lambda}\ -\frac{2}{r_{ij}}\frac{dU}{dr}\right]\mathbf{r}_{ij}^{\lambda}\cdot\mathbf{r}_{ij}^{\lambda} + \sum_i\sum_s\left[\frac{X(r_{is})}{r_{is}^4}\ \mathbf{r}_{is}^{\lambda}\cdot\mathbf{r}_{is}^{\lambda}\ -\frac{2}{r_{is}}\frac{dU}{dr}\right]\mathbf{r}_{is}^{\lambda}\cdot\mathbf{r}_{i}^{\lambda}\right\}, \tag{2.16}$$

where t_P is the so-called pressure coupling/relaxation time and set to be $250\Delta t$ in this project.

As usual, periodic boundary conditions are applied in the *x* and *y* directions to make the simulation systems infinite in the lateral directions. To allow $P_{\|}$ to be controlled instantaneously, the lateral dimensions and coordinates need to be adjusted differentially. In this regard, varying the inter-row spacing between surface atoms or changing the number of rows of surface atoms both have significant shortcomings because the former undesirably alters the crystalline structure of the surfaces and the latter represents finite, not differential, dimension changes. To resolve this difficulty, the periodic boundary conditions are applied only to the confined fluids and the infinite confining surfaces are constructed using a special approach. Specifically, based on the repetitive nature of the crystalline structure possess, surface unit cells can be identified and replicated laterally to form an infinite surface. The smallest unit cell for this purpose is shown in Figure 2.1, where each layer of the Si(111) surface is represented by two Si atoms. Each fluid

particle can now be attributed to a particular surface unit cell characterized by $(n\boldsymbol{a}_1, m\boldsymbol{a}_2)$, where *n*, *m* are integers and $\boldsymbol{a}_1$, $\boldsymbol{a}_2$ are the 2D vectors defining the shape and lateral dimensions of a unit cell (cf. Figure 2.1). By varying *n* and *m* from their central values, a block of surface unit cells centering around a fluid particle can be generated to effectively represent an infinite surface. An additional computational advantage from the method is the neighbor lists for fluid-surface interaction no longer required.

2.3. SIMULATION DETAILS

Throughout the whole project, the total number of confined fluid particles is fixed at 1600, while the number of bigger particles representing $Au_{140}(SC_6H_{13})_{62}$ nanoparticles changes from 0, 1, 3, 5, to 10 in order to investigate the effects of nanoparticle loading/volume fraction. The LJ interaction potential between the solvent cyclohexane particles is truncated and corrected at a center-of-mass cut-off distance of $r_c = 3\sigma_{\text{cyclohexane}}$, or equivalently across a cut-off spacing of $2\sigma_{\text{cyclohexane}}$. For simplicity, the same cut-off spacing of $2\sigma_{\text{cyclohexane}}$ was also used for all the other pair interactions including $Au_{140}(SC_6H_{13})_{62}$–$Au_{140}(SC_6H_{13})_{62}$, $Au_{140}(SC_6H_{13})_{62}$–cyclohexane, $Au_{140}(SC_6H_{13})_{62}$–Si, and cyclohexane-Si. For unlike-pair interactions, the Lorentz-Berthelot mixing rules are used, that is $\varepsilon_{12} = \sqrt{\varepsilon_1 \varepsilon_2}$ and $\sigma_{12} = (\sigma_1 + \sigma_2)/2$.

The integrator for the equations of motion is the fourth-order Gear predictor-corrector algorithm along with a time step of 5 fs. The temperature is set at 300 K and the parallel pressure at 1 atm (0.1 MPa). All confined nanoparticle solutions are first equilibrated to the set temperature and pressure at a separation of 10 nm (100 Å) between

the confining Si (111) surfaces as initial conditions and starting points for continuous compression at a moderate speed of 5 m/s. Practically, to implement continuous compressions at this speed, the surface separation (h) is manually decreased by 0.00025 Å at every time step ($\Delta t = 5$ fs). For precaution, additional 25000 Δt' s were spent to re-equilibrate the simulation systems before compression and every production run in this project.

3. RESULTS AND DISCUSSION

3.1 PC-BASED MOLECULAR SIMULATIONS

The same simulation code and initial conditions were used to run simulations of continuous compression of confined nanoparticle-alkane solutions having 0, 1, 3, 5, and 10 nanoparticles were run on IBM RISC/6000 workstations and on a dual-core Dell XPS computer installed with Linux. As demonstrated in Figure 3.1 where the case with 5 nanoparticles is shown, the simulation results from the two systems are not identical but statistically and physically equivalent. This was expected because after a large number of iterations, the differences in computer architecture, operation system, and compiler will cause numerical calculations to deviate on different computer systems. Nevertheless, the physics and the simulated properties are not altered, which is of critical importance. It should be emphasized here that with a relatively finite number of fluid particles (e.g. 1600), the pressure of a MD simulation cannot be precisely controlled to any specific value, which is very unlike temperature control. In fact, it is natural for MD simulations to have pressure fluctuations whose magnitudes are on the order of several tens to several hundreds atm. From this point of view, the method of parallel pressure ($P_{\parallel}$) control explained in the previous section works very well and the calculated perpendicular pressures ($P_{\perp}$) from the two computer systems also agree well.

It should be mentioned that with multiple cores and free parallel computing software such as Open MPI, the new generation of PC's are readily capable of parallel computing. We have tested serial and parallel MD simulations on the Dell XPS computer and

interestingly found the results to be similar to what are demonstrated here, numerically not identical but statistically and physically equivalent.

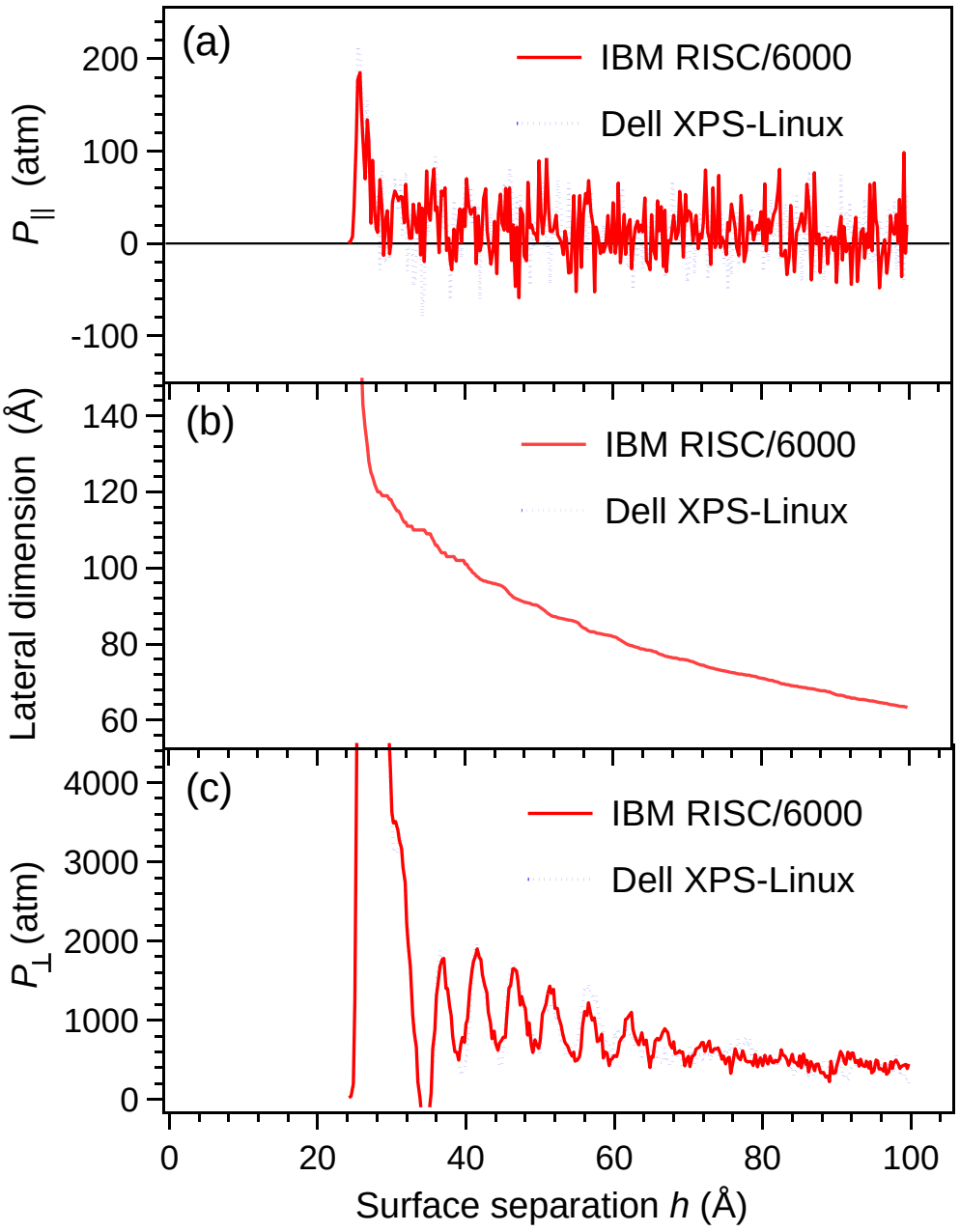

Figure 3.1. Instantaneous (a) parallel pressure, (b) lateral dimension, and (c) perpendicular pressure as a function surface separation during continuous compression.

3.2 CONTINUOUS COMPRESSION

Figure 3.2 shows the simulation results from continuous compression of different nanoparticles solutions. The data are instantaneous properties calculated and recorded

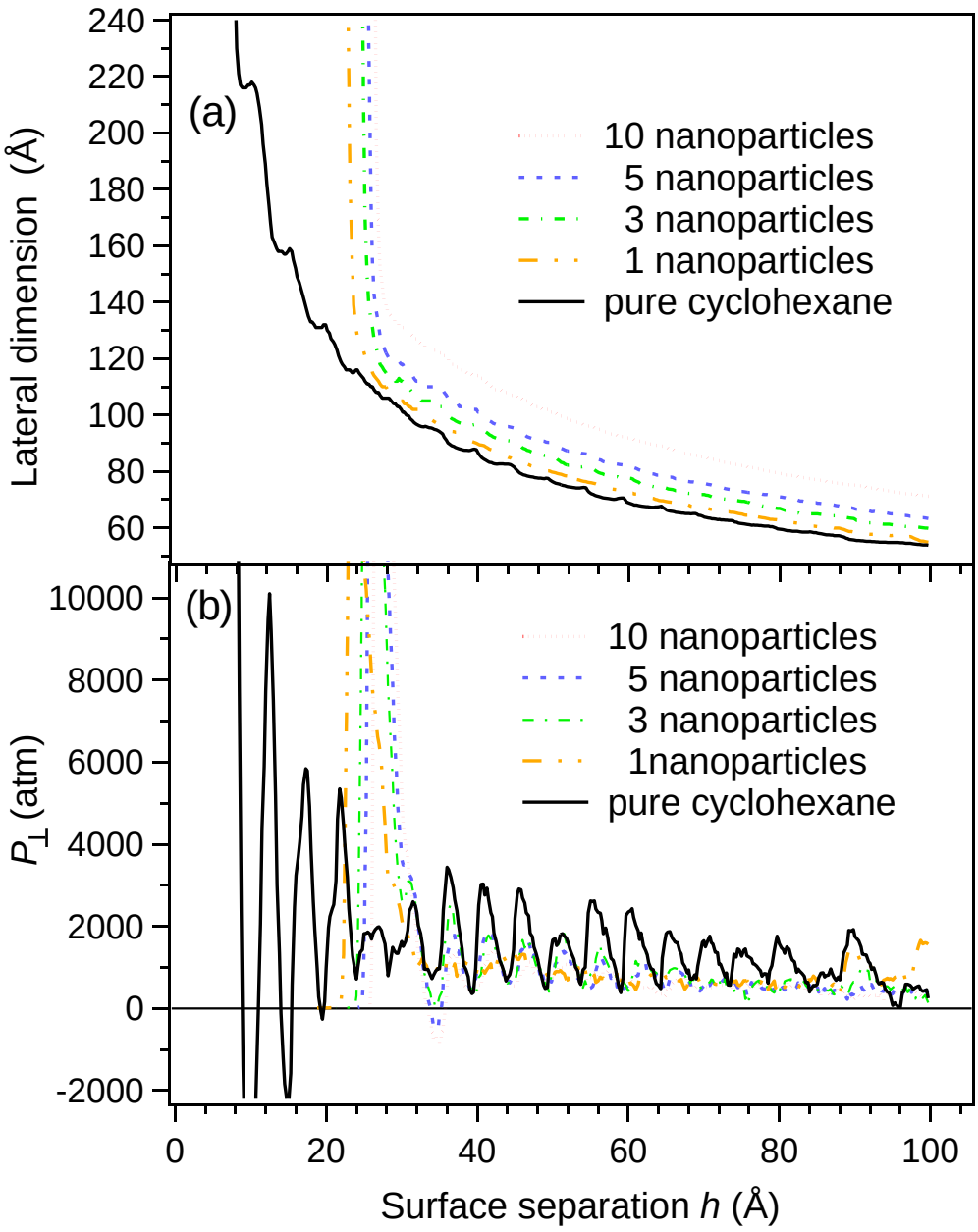

Figure 3.2. Instantaneous (a) lateral dimension and (b) perpendicular pressure as a function surface separation during continuous compression for different solutions.

every 1000 Δt's. Exhibiting the strongest force oscillations [cf. Fig. 3.2(b)] is the pure cyclohexane fluid in nanoconfinement that can be understood to be most layered among all solutions studied. A closer examination of the force oscillations reveals that the periodicity is about the same as the cyclohexane molecular diameter, 5.466 Å. The connection between the surface force and the fluid configuration can be analyzed by comparing Figures 3.2 (a) and (b) together. It can be easily seen that the lateral dimension of the confined cyclohexane under continuous compression undergoes step-like expansion and the width of the steps is the same as the periodicity of the force oscillations. More specifically, minimum surface forces occur at separations that correspond to completion of step-like expansion and more disordered, less layered configurations. When compression continues, the less layered confined fluid are squeezed to become more layered and the surface force builds up again until the confined fluid could not sustain the strong pressure any more and sudden expansion occurs in the lateral directions.

When nanoparticles are added to the confined alkane fluid, they significantly reduce the magnitudes of the step-like expansion and force oscillations. In addition, the bigger nanoparticles can now resist compression and strong perpendicular pressure at larger separations that are about the same as the size of the nanoparticles [cf. Fig. 3.2(a)]. These behaviors signify better nanotribological properties that could be provided by the presence of alkanethiol-capped nanoparticles in alkane/oil solutions.

Since the nanoparticles have much bigger size and stronger interaction, there may exist a possibility that they could aggregate together in the solutions and this possibility could get enhanced by the nanoscale confinement. In order to obtain an answer and more

insight, we examine the top-down views and side views of the confined nanoparticle solutions. Shown in Figure 3.3 are the representative case that has 10 nanoparticles and pure cyclohexane for comparison. It can be clearly seen that at large separations [cf. Fig. 3.3 (c)], nanoparticles are well dispersed in the confined fluid without either forming clusters or aggregating towards the confining surfaces and they remain well dispersed as the confined fluid film becomes thinner and thinner. Even when the nanoparticles are the confined fluid film becomes thinner and thinner. Even when the nanoparticles are encapsulated and touch both confining surfaces, they are still surrounded by solvent

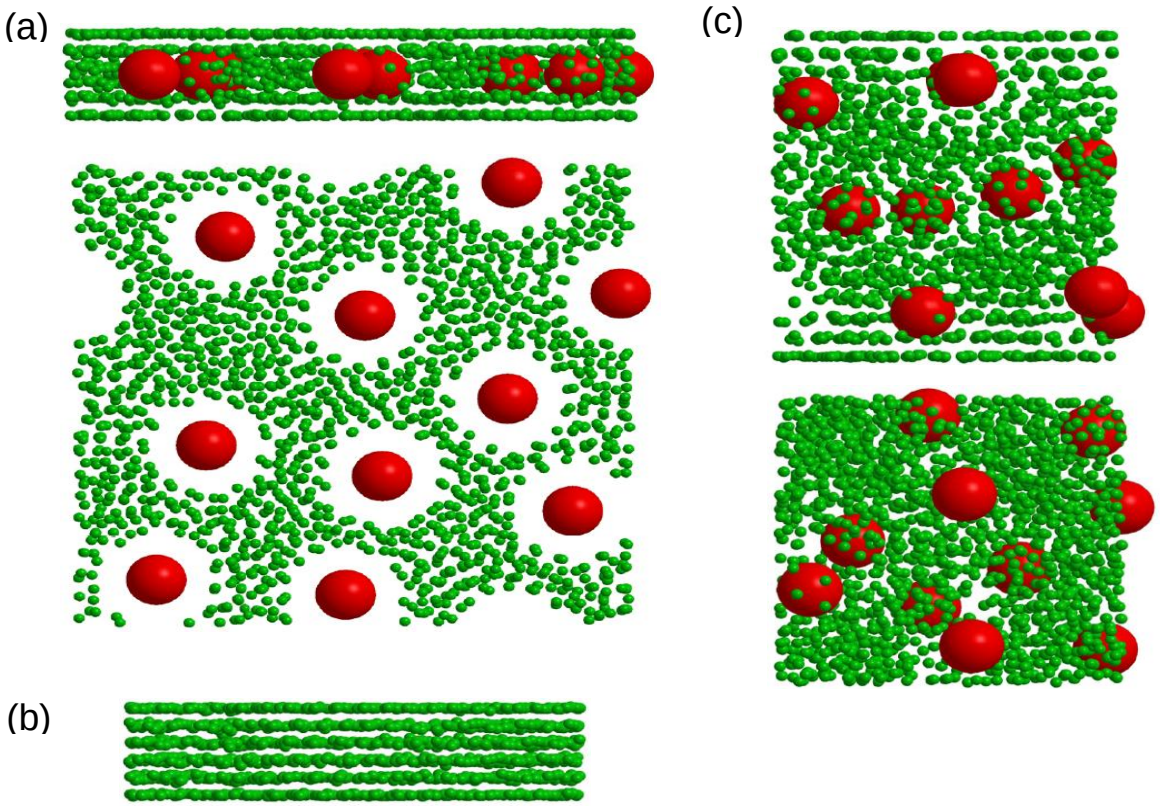

Figure 3.3. (a) Side and top-down views of the nanoconfined cyclohexane (green beads) solution containing 10 nanoparticles (red beads) at $h = 31.25$ Å, (b) side view of confined cyclohexane at $h = 31.25$ Å, and (c) side and top-down views of the nanoconfined cyclohexane solution containing 10 nanoparticles at $h = 93.75$ Å.

molecules [cf. Fig. 3.3(a)]. These observations indicate a proper affinity between the nanoparticles and the solvent. As can bee seen in Figure 3.3, solvent molecules within certain distance from the nanoparticles are under the influence of nanoparticle interactions which lessens the layering effect of nanoconfinement. However, outside this influence range, solvent molecules are still largely layered.

3.3 NANOCONFINED NANOPARTICLE SOLUTIONS UNDER SHEAR

Form the results presented and discussed above, adding nanoparticles appears to be able to improve the nanotribological properties of alkane-type lubricants. To further validate this point, the nanoparticle solutions are sheared by sliding the upper confining surface in the +*x* direction at a speed of $v_x = 0$ m/s, 1 m/s, or 10 m/s while keeping the lower surface stationary at three selected separations $h = 31.25$ Å, 56.25 Å, and 93.75 Å. The resultant nonzero shear rates, $\dot{\gamma} = v_x / h$, depend on both the sliding speed and surface separation and range from 1.07×10^8 sec^{-1} to 3.2×10^9 sec^{-1}. The stationary confined systems with zero shear rate are simulated for comparison purposes.

We first examine the density profiles of cyclohexane solvent molecules under various conditions. As shown in Figure 3.4, the density profile depends quite strongly on the number of nanoparticles. Without the presence of nanoparticles, the symmetric cyclohexane molecules can form well-layered configurations in nanoconfinement, which can be disrupted to become less layered when nanoparticles are present. On the other hand, the density profile is a very weak function of shear rate and exhibits only very insignificant changes within the explored shear rate range.

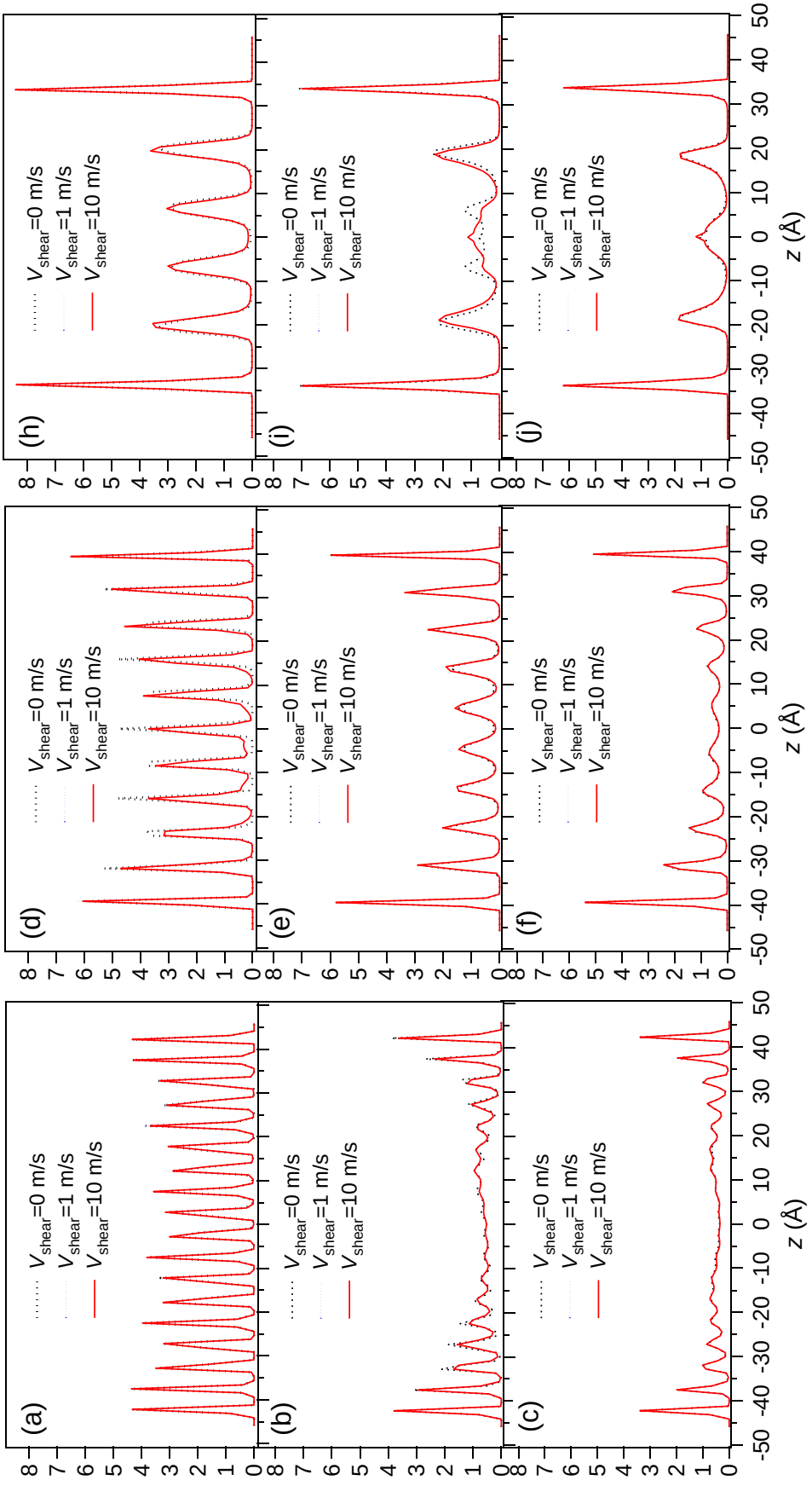

Figure 3.4. Profiles of cyclohexane number density where (a)~(c) at h = 93.75 Å, (d)~(f) at h = 56.25, and (h)~(j) at h = 31.25 Å; and (a), (d), (h): 0 nanoparticle, (b), (e), (i): 5 nanoparticles, and (c), (f), (j): 10 nanoparticles.

Shear stress could be imparted into the nanoconfined fluids in different manners. Another major method is to apply a constant shear force to a confining surfacewhich is assigned a proper artificial mass so that it has its own equations of motion to determine its displacement and sliding velocity. While this method allows a constant shear stress, the shear rate is variable. In contrast, the method adopted in this work can maintain a constant shear rate but the shear stress becomes variable. Nevertheless, these two methods can be made equivalent by taking time averages of shear stress and shear rate. In our simulations, the applied shear generates a nonzero drag force on the confined fluid in the *x* direction, f_x, which would become noisy fluctuations around zero under no shear condition. Shear stress can be calculated by dividing the drag force f_x by surface area of the confining surface. Figure 3.5 shows the calculated shear stress for the confined fluid film with 10 nanoparticles at $h = 56.25$ Å. The time averaged shear stress is 3.71 MPa for $v_x = 10$ m/s corresponding to $\dot{\gamma} = 1.78 \times 10^9$ sec^{-1} and 0.87 MPa for $v_x = 1$ m/s corresponding to $\dot{\gamma} = 1.78 \times 10^8$ sec^{-1}. As a result, the apparent viscosity is 0.021 Pa·s and 0.0049 Pa·s, respectively. We have performed such calculations for different cases considered in this project and summarized the results in Table 3.1. The calculated viscosities appear to be in good agreement with those from similar simulation studies [60]. Most importantly, at both sliding speeds and all three separations, the presence of nanoparticles in the confined thin films does help reduce noticeably the viscosity or equivalently shear stress and frictional force. We can thus conclude that nanoparticle-containing solutions have great potential to be good lubricants for nanotribology.

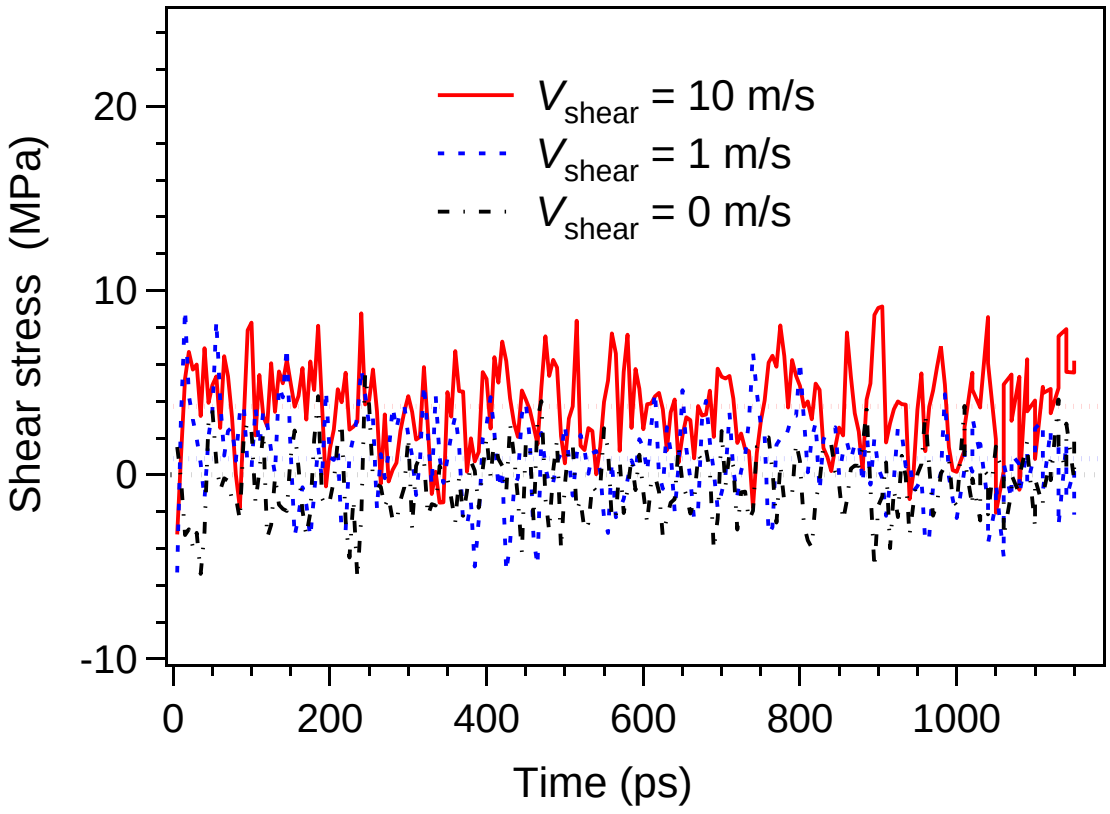

Figure 3.5. Shear stress on the confined fluid having 10 nanoparticles at $h = 56.25$ Å caused by different sliding speeds.

Table 3.1. Apparent shear viscosity of nanoparticle-cyclohexane solutions under different surface separation and sliding speeds.

No. of nanoparticles	$h =$ 93.75 Å		$h =$ 65.25 Å		$h =$ 31.25 Å	
	$v_x = 1$ m/s	10 m/s	1 m/s	10 m/s	1 m/s	10 m/s
0	0.0385	0.0444	0.00866	0.0255	0.00716	0.0234
5	0.00368	0.0136	0.00726	0.00973	0.00384	0.00276
10	0.000677	0.000671	0.00491	0.0209	0.00472	0.00675

Despite their sizes, alkanethiol-capped nanoparticles have been shown by simulation studies to have fluid-like diffusivity [48]. Since the potential models used in this work are atomistically based, it would be of relevance and value to compute both the nanoparticles and cyclohexane diffusivities. To this end, the following Einstein relation is used,

$$D_{\parallel} = \lim_{t\to\infty} \frac{\left\langle \mathbf{r}_{\parallel}(t) - \mathbf{r}_{\parallel}(0)^{2} \right\rangle}{4t} , \tag{3.1}$$

where $\left\langle \mathbf{r}_{\parallel}(t) - \mathbf{r}_{\parallel}(0)^{2} \right\rangle$ is the ensemble-averaged mean-square displacement in the lateral x and y directions. It should be self-evident that a confined fluid has zero net displacement/diffusivity in the z direction. By calculating and plotting the mean-square displacement using the x and y components, the limiting slope after sufficiently long times is equal to $4D_{\parallel}$. Shown in Figure 3.6 is a plot of the nanoparticles mean square displacement for the 10-nanoparticle confined film at h = 93.75 Å with zero shear. All the calculated diffusivities are collected in Table 3.2. In general, nanoparticles in nanoconfinement have diffusivity noticeably lower than that of typical liquids. In particular at small separation when nanoparticles directly touch both confining surfaces, their mobility is virtually zero. Also, as can be expected, sliding speed shears the confined fluid components and increases their mobility.

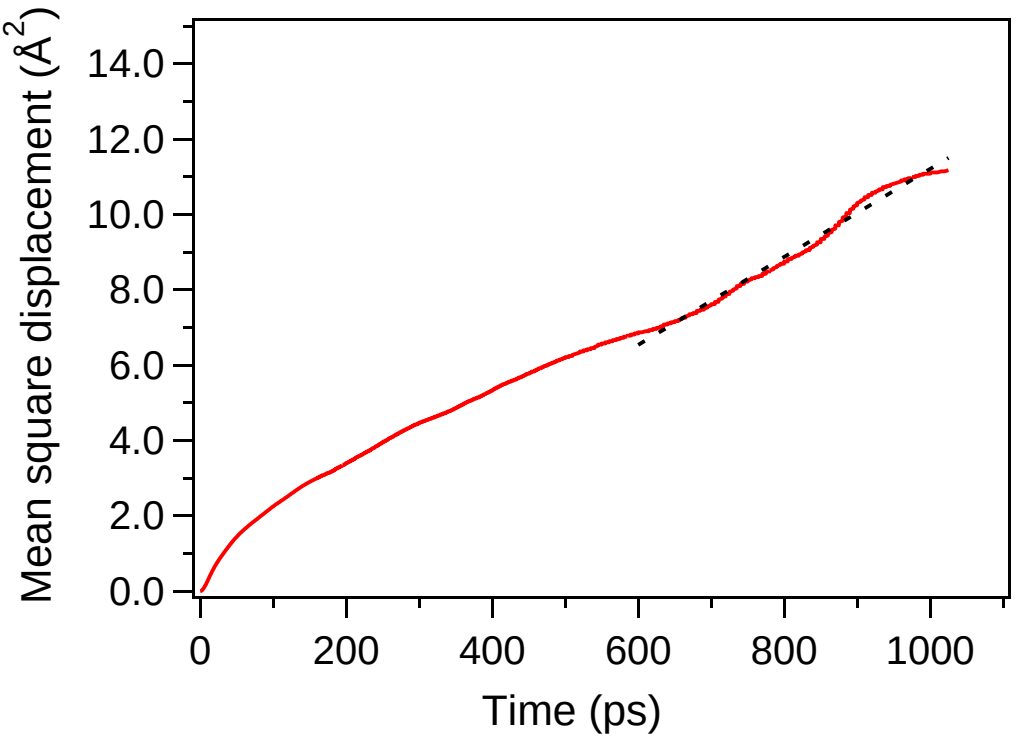

Figure 3.6. A plot of mean square displacement where the fitted dashed line is used to determine the diffusivity.

Table 3.2. Diffusion coefficients $D\times10^{10}$ m^2/s.

No. of nanoparticles	h = 93.75 Å					
	v_x = 0 m/s		v_x = 1 m/s		v_x = 10 m/s	
	cycloC_6	nanoparticle	cycloC_6	nanoparticle	cycloC_6	nanoparticle
0	0.27		1.08		25.72	
5	11.14	0.48	15.21	3.41	62.47	64.02
10	15.23	0.54	14.87	0.63	21.51	7.74
No. of nanoparticles	h = 65.25Å					
	v_x = 0 m/s		v_x = 1 m/s		v_x = 10 m/s	
	cycloC_6	nanoparticle	cycloC_6	nanoparticle	cycloC_6	nanoparticle
0	0.12		2.24		10.96	
5	2.09	0.02	2.51	0.15	76.18	73.85
10	6.27	0.42	7.02	0.75	79.86	79.35
No. of nanoparticles	h = 31.25Å					
	v_x = 0 m/s		v_x = 1 m/s		v_x = 10 m/s	
	cycloC_6	nanoparticle	cycloC_6	nanoparticle	cycloC_6	nanoparticle
0	0.12		1.34		50.19	
5	0.84	0.06	1.97	1.17	56.17	55.87
10	1.22	0.02	2.48	1.43	32.95	32.18

4. CONCLUSIONS

Current advances in micro and nano-electromechanical devices (MEMS and NEMS), micro and nanofluidic systems, and other nanotechnology areas constantly involve the tribological issues of friction, wear, and lubrication at the nanometer i.e. molecular scale. Despite recent progresses, major challenges still loom over nanometer-thin lubricant films, partly due to the unusual and undesirable tribological properties of typical lubricant molecules forming layered configurations in nanoconfinement and partly due to the limitation of experimental techniques and conventional theories. In this study, solutions of alkanethiol-capped nanoparticles in alkane are examined using molecular dynamics simulations for their nanotribological potential based on the hypothesis that fluid molecules of very different sizes may interrupt each other's layering tendency to result in less or non-layered configurations and provide better lubrication for nanodevices. An effective nanoparticle-nanoparticle pair potential based on previous atomistic approach is used and the temperature and parallel pressure are controlled in place of chemical potential for defining thermodynamic state. When compressed, the confined nanoparticle-containing alkane films generate reduced oscillations in perpendicular forces and smoother expansion in lateral dimensions. This indicate lesser extent of layering due to the presence of much bigger nanoparticles, which was confirmed by the density profiles obtained in his study. Further examination reveal that the nanoparticles are well dispersed by the cyclohexane solvent molecules throughout all separations. This means that the nanoparticles will not tend to form clusters or aggregate towards the confining surfaces, which is important for the stability of the nanoparticle solutions as nanotribological lubricant. When sheared by a sliding surface, the confined

fluids tend to move in the same parallel manner so that their density profiles remain virtually unchanged. The shear stress imposed by the sliding surface has also been calculated and then used to estimate the apparent viscosity. It is found that greater shear stress does result from faster sliding but not proportionally. More importantly, the presence of nanoparticles in the lubricant films reduces the shear stress and thereby apparent viscosity noticeably. This effect is particularly evident under relatively large sliding speed and large surface separations. In addition, the nanoparticles exhibit lower diffusivity in nanoconfinement than typical fluids and their mobility can be enhanced by shearing. In summary, this study has demonstrated the great potential of nanoparticle-containing solutions as an improved lubricant for moving nanodevices and nanostructures.

Additionally, simulations have been run on regular PC's and physically and statistically equivalent results have been obtained. This points out a future direction for similar simulation studies, that is, using multi-core PC's and freely available compilers and parallel computing software is the most efficient and most cost-effective way. With parallel computing, much larger simulation system can be employed and semi-continuous variation of nanoparticles loading/volume fractions and more systematic knowledge can be achieved in simulations.

APPENDIX

FORTRAN77 code for the Molecular Dynamics Simulation.

```
c      PROGRAM real
c
       IMPLICIT real*8 (a-h,o-z),integer*4 (i-n)
       include 'ljmix'

C      POTENTIAL PARAMETERS
C      ____________________________________________________
C
C      Three basic quantities for non-dimensionalization
C      based on parameters for cyclohexane(l) as solvent
C      J. Mol. Liq. 120, 63 (2005)
C      ____________________________________________________

       sigma  = 5.4660d-10 ! meter
       epsi   = 484.00d0*R ! J/mol
       unitm  = 0.084160d0 ! kg/mol
C      ____________________
C
C      System parameters
C      Note that
C        1 fs  = 0.00020002
C        1 m/s = 0.00457325
C        1 Pa  = 2.444e-8
C      ____________________

       dt    = 5.00d-15                    ! time step: second
       h     = dt*dsqrt(epsi/unitm)/sigma ! non-dimensionalized dt
       temp  = tset*R/epsi
       uconv = dsqrt(unitm/epsi)          ! u*=u*uconv
       pconv = sigma*sigma*sigma*6.0220d23/epsi  !0.244396E-07
       pset  = (ppara*101325)*pconv
       pi    = dacos(-1.0d0)
C      ____________________________________________________________________
C
C      Lennard-Jones interaction parameters and long-range corrections
C      sa:   sa-sa (surface atom-surface atom)
C      gsgs: solvent-solvent (smaller fluid particles)
C      gssa: solvent-surface
c      gbgb: nanoparticle-nanoparticle (bigger fluid particles)
c      gbsa: nanoparticle-surface
C      lrc:  long-range corrections
C      ____________________________________________________________________
C      --------------------------------
C      Stationary confining Si surfaces
C      --------------------------------
       al    = 5.43d-10/sigma     ! Si diamond structure lattice const.
       x_L   = al*sqrt(2.0)/2.0d0 ! 3.84A
       y_L   = x_L*sqrt(3.0)      ! 6.65A
```

```
      z_L   = (al/2.0d0)*sqrt(3.0)/2.0d0  ! Si-Si bond length,2.35A

      sa_sig    = 3.8260000d-10  ! sa-sa
      sa_eps    = 202.450d0*R    ! sa-sa
      sam       = 0.0280855d0    ! kg/mol
      sam       = sam/unitm
      epssa     = sa_eps/epsi
      eps4sa    = 4.00d0*epssa
      eps24sa   = 24.0d0*epssa
      sigsa     = sa_sig/sigma
      sig2sa    = sigsa*sigsa

      zk    = pi/(sqrt(2.0d0/3.0d0)*1.09d0)
C     -------------------------------------
C     Like pair of smaller solvent particles
C     -------------------------------------
      gsgs_sig  = 5.4660000d-10  ! sfa-sfa
      gsgs_eps  = 484.00d0*R     ! sfa-sfa
      sfam      = 0.084160d0     ! kg/mol
      sfam      = sfam/unitm
      epssfsf   = gsgs_eps/epsi
      eps4sfsf  = 4.00d0*epssfsf
      eps24sfsf = 24.0d0*epssfsf
      sigsfsf   = gsgs_sig/sigma
      sig2sfsf  = sigsfsf*sigsfsf
      rcutsfsf  = sigsfsf+2.0d0*sigsfsf    ! potential cut-off distance
      rcut2sfsf = rcutsfsf*rcutsfsf
      drneig    = 0.30d0                    ! neighbor list buffer
      rnbl2sfsf = (rcutsfsf+2.0d0*drneig)**2   ! distance for neighbor
C                                               !list
      rlrc2     = sig2sfsf/rcut2sfsf
      rlrc6     = rlrc2*rlrc2*rlrc2
      rlrc12    = rlrc6*rlrc6
      vcsfsf    = (13.0d0*rlrc12 - 7.0d0*rlrc6)
      fcsfsf    = (2.0d0*rlrc12 - rlrc6)/rcutsfsf
      fcvsfsf   = 6.0d0*fcsfsf
C     ------------------------------------
C     Like pair of bigger nanoparticles
C     resembling Au140(SC6H13)62
C     ------------------------------------
      gbgb_sig  = 29.000000d-10  ! bfa-bfa
      gbgb_eps  = 22.37*gsgs_eps ! bfa-bfa
      bfam      = 34.844000d0    ! kg/mol
      bfam      = bfam/unitm
      epsbfbf   = gbgb_eps/epsi
      eps4bfbf  = 4.00d0*epsbfbf
      eps24bfbf = 24.0d0*epsbfbf
      sigbfbf   = gbgb_sig/sigma
      sig2bfbf  = sigbfbf*sigbfbf
      rcutbfbf  = sigbfbf+2.0d0*sigsfsf    ! potential cut-off distance
      rcut2bfbf = rcutbfbf*rcutbfbf
      drnblbfbf = 0.30d0                        ! neighbor list buffer
      rnbl2bfbf = (rcutbfbf+2.0d0*drnblbfbf)**2  ! distance for
      rlrc2     = sig2bfbf/rcut2bfbf              ! neighbor list
      rlrc6     = rlrc2*rlrc2*rlrc2
      rlrc12    = rlrc6*rlrc6
      vcbfbf    = (13.0d0*rlrc12 - 7.0d0*rlrc6)
```

```
      fcbfbf    = (2.0d0*rlrc12 - rlrc6)/rcutbfbf
      fcvbfbf   = 6.0d0*fcbfbf
C     ------------------------------------
C     unlike pair of solvent-nanoparticle
C     ------------------------------------
      epssfbf   = dsqrt(epsbfbf*epssfsf)
      eps4sfbf  = 4.00d0*epssfbf
      eps24sfbf = 24.0d0*epssfbf
      sigsfbf   = 0.50d0*(sigbfbf+sigsfsf)
      sig2sfbf  = sigsfbf*sigsfbf
      rcutsfbf  = sigsfbf+2.0d0*sigsfsf    ! potential cut-off distance
      rcut2sfbf = rcutsfbf*rcutsfbf
      drnblsfbf = 0.30d0                   ! neighbor list buffer
      rnbl2sfbf = (rcutsfbf+2.0d0*drnblsfbf)**2  ! distance for
      rlrc2     = sig2sfbf/rcut2sfbf              ! neighbor list
      rlrc6     = rlrc2*rlrc2*rlrc2
      rlrc12    = rlrc6*rlrc6
      vcsfbf    = (13.0d0*rlrc12 - 7.0d0*rlrc6)
      fcsfbf    = (2.0d0*rlrc12 - rlrc6)/rcutsfbf
      fcvsfbf   = 6.0d0*fcsfbf
C     ------------------------------------
C     unlike pair of solvent-surface
C     ------------------------------------
      epssfsa   = dsqrt(epssfsf*epssa)
      eps4sfsa  = 4.00d0*epssfsa
      eps24sfsa = 24.0d0*epssfsa
      sigsfsa   = 0.50d0*(sigsfsf+sigsa)
      sig2sfsa  = sigsfsa*sigsfsa
      rcutsfsa  = sigsfsa+2.0d0*sigsfsf    ! potential cut-off distance
      rcut2sfsa = rcutsfsa*rcutsfsa
      drnblsfsa = 0.30d0                   ! neighbor list buffer
      rnbl2sfsa = (rcutsfsa+2.0d0*drnblsfsa)**2  ! distance for
      rlrc2     = sig2sfsa/rcut2sfsa       ! neighbor list
      rlrc6     = rlrc2*rlrc2*rlrc2
      rlrc12    = rlrc6*rlrc6
      vcsfsa    = (13.0d0*rlrc12 - 7.0d0*rlrc6)
      fcsfsa    = (2.0d0*rlrc12 - rlrc6)/rcutsfsa
      fcvsfsa   = 6.0d0*fcsfsa
C     ------------------------------------
C     unlike pair of Rnanoparticle-surface
C     ------------------------------------
      epsbfsa   = dsqrt(epsbfbf*epssa)
      eps4bfsa  = 4.00d0*epsbfsa
      eps24bfsa = 24.0d0*epsbfsa
      sigbfsa   = 0.50d0*(sigbfbf+sigsa)
      sig2bfsa  = sigbfsa*sigbfsa
      rcutbfsa  = sigbfsa+2.0d0*sigsfsf    ! potential cut-off distance
      rcut2bfsa = rcutbfsa*rcutbfsa
      drnblbfsa = 0.30d0                   ! neighbor list buffer
      rnbl2bfsa = (rcutbfsa+2.0d0*drnblbfsa)**2  ! distance for
      rlrc2     = sig2bfsa/rcut2bfsa              ! neighbor list
      rlrc6     = rlrc2*rlrc2*rlrc2
      rlrc12    = rlrc6*rlrc6
      vcbfsa    = (13.0d0*rlrc12 - 7.0d0*rlrc6)
      fcbfsa    = (2.0d0*rlrc12 - rlrc6)/rcutbfsa
      fcvbfsa   = 6.0d0*fcbfsa
```

```
c     ____________
C
C     MAIN PROGRAM
C     ____________

      CALL simulation

      STOP
      END


      SUBROUTINE simulation
      IMPLICIT real*8 (a-h,o-z),integer*4 (i-n)
      include 'ljmix'

      DIMENSION  diff(3,nadmol),disp(2,nadmol)
      DIMENSION  z1(nadmol),z2(nadmol),z3(nadmol),z4(nadmol)
      DIMENSION  vz1(nadmol),vz2(nadmol),vz3(nadmol),vz4(nadmol)
      DIMENSION  x1(nadmol),x2(nadmol),x3(nadmol),x4(nadmol)
      DIMENSION  vx1(nadmol),vx2(nadmol),vx3(nadmol),vx4(nadmol)
      DIMENSION  y1(nadmol),y2(nadmol),y3(nadmol),y4(nadmol)
      DIMENSION  vy1(nadmol),vy2(nadmol),vy3(nadmol),vy4(nadmol)
      DIMENSION  densf(nbin),denbf(nbin),adist(nbin),bdist(nbin)
      LOGICAL    disptest

      OPEN(1,file='5np.sh_m',status='unknown',form='formatted')
      OPEN(2,file='5np.dat_m0',status='unknown',form='formatted')
      OPEN(3,file='5np.rec_m0',status='unknown',form='formatted')

      vcmx     = 0.0d0
      vcmy     = 0.0d0
      vcmz     = 0.0d0
      systemas = 0.0d0
      DO m=1,nsfmol
        read(1,100)x0(m),y0(m),z0(m),vx0(m),vy0(m),vz0(m)
        systemas = systemas + sfam
C       ------------------------------
C       Call gauss to generate initial
C       velocities if needed
C       ------------------------------
c       Call gauss(v1,v2,v3,temp)
c       vx0(m) = v1
c       vy0(m) = v2
c       vz0(m) = v3
        vcmx = vcmx + vx0(m)*sfam
        vcmy = vcmy + vy0(m)*sfam
        vcmz = vcmz + vz0(m)*sfam
      ENDDO
      DO m=nsfmol+1,nadmol
        read(1,100)x0(m),y0(m),z0(m),vx0(m),vy0(m),vz0(m)
        systemas = systemas + bfam
        vcmx = vcmx + vx0(m)*bfam
        vcmy = vcmy + vy0(m)*bfam
        vcmz = vcmz + vz0(m)*bfam
      ENDDO
      read(1,*)bLx0,bLy0,hz0
```

```
      vcmx = vcmx/systemas
      vcmy = vcmy/systemas
      vcmz = vcmz/systemas
      vsumx = 0.0d0
      vsumy = 0.0d0
      vsumz = 0.0d0
      DO m=1,nsfmol
        vx0(m) = vx0(m) - vcmx
        vy0(m) = vy0(m) - vcmy
        vz0(m) = vz0(m) - vcmz
        vsumx  = vsumx + vx0(m)*vx0(m)*sfam
        vsumy  = vsumy + vy0(m)*vy0(m)*sfam
        vsumz  = vsumz + vz0(m)*vz0(m)*sfam
      ENDDO
      DO m=nsfmol+1,nadmol
        vx0(m) = vx0(m) - vcmx
        vy0(m) = vy0(m) - vcmy
        vz0(m) = vz0(m) - vcmz
        vsumx  = vsumx + vx0(m)*vx0(m)*bfam
        vsumy  = vsumy + vy0(m)*vy0(m)*bfam
        vsumz  = vsumz + vz0(m)*vz0(m)*bfam
      ENDDO
      etuax = vsumx/nadmol
      etuay = vsumy/nadmol
      etuaz = vsumz/nadmol
      tratx = sqrt(temp/etuax)
      traty = sqrt(temp/etuay)
      tratz = sqrt(temp/etuaz)
      do m=1,nadmol
        vx0(m) = vx0(m)*tratx
        vy0(m) = vy0(m)*traty
        vz0(m) = vz0(m)*tratz
      enddo
 100  format(6(1x,e11.5))

C     ------------------------------------------
C     Determine (x,y,z) coordnates for surface atoms
C     ------------------------------------------
      rsold(1) = -0.50d0*hz0  !! lower surface z position
      rsold(2) = rsold(1) - z_L/3.0d0
      rsold(3) = rsold(2) - z_L
      rsold(4) = rsold(3) - z_L/3.0d0
      rsold(5) = rsold(4) - z_L
      rsold(6) = rsold(5) - z_L/3.0d0
      rsold(7) = rsold(6) - z_L
      rsold(8) = rsold(7) - z_L/3.0d0
      DO i=1,layers
        rsold(i+layers)=-rsold(i)
      ENDDO
C     ------------------------------------------
C     Generate (x,y) coordnates for surface atoms
C     that represent the Si_diamond(111) surface.
C     ------------------------------------------

      Call setup_sa_position

C     ------------------------------------------------------
```

```
C     Options: istart=1: separation fixed at the input value
C              istart=2: separation adjustment is activated
C              istart=3: production run at a fixed separation
C     ------------------------------------------------------

      istart = 2

C     ------------------------------------------------
C     hzi: initial separation (to be adjusted)
C     hzf: final separation (after certain no. of steps)
C     ------------------------------------------------
      hzi   = hz0
      hzf   = 10.0d0/5.4660d0   ! use desired value
C     ------------------------------------------------------------
C     deltz: displacement of the surfaces per time step due to
C            continuous compression.
C     deltx: displacement of the upper surface per time step
C            for continuous shearing (stationary lower surface)
C     ------------------------------------------------------------
c     deltz = 0.50d0*(hzf - hzi)/nsimut
      deltz = -0.50d0*(unorm*uconv)*h
      deltx = (upara*uconv)*h
      dxtot = 0.0d0
C     ------------------------------------------------
C     Gear predictor-corrector integration coefficients
C     ------------------------------------------------
      f01 = 251.0d0/720.0d0
      f21 = 11.0d0/12.0d0
      f31 = 1.0d0/3.0d0
      f41 = 1.0d0/24.0d0
C     -------------------------------------------
C     Pressure and thermal bath coupling constants
C     Thermal bath: Nose-Hoover thermostat
c     Pressure bath: Berendsen's loose coupling
C     -------------------------------------------
      tp       = 250*h
      ttnh     = 45.0d0
C     -------------------------------------------
C     Bin sizes for analyzing various distributions
C     -------------------------------------------
      ncen = (nbin+1)/2
      dz   = hz0/(nbin+1)

 200  nstp   = 0
      icntpt = 0
      icnt   = 0
      index  = 0

      totint   = 0.0d0
      totmol   = 0.0d0
      tmolsum  = 0.0d0
      tmol2sum = 0.0d0
      bLxtot   = 0.0d0
      bLxt2    = 0.0d0
      bLxt4    = 0.0d0
      skztot   = 0.0d0
      schange  = 0.0d0
```

```
ppsfsftot = 0.0d0
ppsfbftot = 0.0d0
ppsfsatot = 0.0d0
ppbfbftot = 0.0d0
ppbfsatot = 0.0d0
pzsfsftot = 0.0d0
pzsfbftot = 0.0d0
pzsfsatot = 0.0d0
pzbfbftot = 0.0d0
pzbfsatot = 0.0d0

pxytot   = 0.0d0
pxy2     = 0.0d0
pztot    = 0.0d0
pz2      = 0.0d0
fsoltot  = 0.0d0
fsol2    = 0.0d0

fzsftot  = 0.0d0
fzbftot  = 0.0d0
fxxdtot  = 0.0d0
fxxutot  = 0.0d0

DO m=1,nadmol
  x1(m)  = 0.0d0
  x2(m)  = 0.0d0
  x3(m)  = 0.0d0
  x4(m)  = 0.0d0
  vx1(m) = 0.0d0
  vx2(m) = 0.0d0
  vx3(m) = 0.0d0
  vx4(m) = 0.0d0
  y1(m)  = 0.0d0
  y2(m)  = 0.0d0
  y3(m)  = 0.0d0
  y4(m)  = 0.0d0
  vy1(m) = 0.0d0
  vy2(m) = 0.0d0
  vy3(m) = 0.0d0
  vy4(m) = 0.0d0
  z1(m)  = 0.0d0
  z2(m)  = 0.0d0
  z3(m)  = 0.0d0
  z4(m)  = 0.0d0
  vz1(m) = 0.0d0
  vz2(m) = 0.0d0
  vz3(m) = 0.0d0
  vz4(m) = 0.0d0
  DO i=1,3
    diff(i,m)  = 0.0d0
ENDDO
  DO i=1,2
    disp(i,m)  = 0.0d0
  ENDDO
  DO k=1,itau
    rm2(1,m,k) = 0.0d0
```

```
          rm2(2,m,k) = 0.0d0
        ENDDO
      ENDDO

      DO n=1,nbin
        densf(n) = 0.0d0
        denbf(n) = 0.0d0
      ENDDO
c     ____________________________________
c
c     INITIATE NEIGHBOR LISTS and use
c     disptest=.true. Update neighbor lists
c     ____________________________________

      CALL fluid_neighbor_list

      disptest=.false.
c     ____________________________
c
c     MAIN LOOP over nsimut tme steps
c     ____________________________

      DO while (nstp.le.nsimut)

      nstp   = nstp+1
        icntpt = icntpt+1

        IF (disptest.eqv..true.) then   !! update neighbor lists
c         write(*,*)'nstp=',nstp,'--- update neighbor lists ---'
          CALL fluid_neighbor_list
          disptest=.false.
          DO m=1,nadmol
            diff(1,m)=0.0d0
            diff(2,m)=0.0d0
            diff(3,m)=0.0d0
          ENDDO
          schange=0.0d0
        ENDIF
c       ____________________________________________________
c
c       GEAR'S PREDICTOR PHASE : truncated power series
C       to predict coordinates and velocities
c       ____________________________________________________

        vx2sum = 0.0d0
        vy2sum = 0.0d0
        vz2sum = 0.0d0
        v13sum = 0.0d0
        v23sum = 0.0d0
        vxytemp= 0.0d0
        vztemp = 0.0d0
        DO m=1,nsfmol

          vxsq   = vx0(m)*vx0(m)
          vysq   = vy0(m)*vy0(m)
          vzsq   = vz0(m)*vz0(m)
          vx2sum = vx2sum+vxsq
```

```
      vy2sum = vy2sum+vysq
      vz2sum = vz2sum+vzsq
      v13sum = v13sum+vx0(m)*vz0(m)
      v23sum = v23sum+vy0(m)*vz0(m)

      xold(m)=x0(m)
      x0(m)=x0(m)+       x1(m)+       x2(m)+       x3(m)+x4(m)
      x1(m)=x1(m)+2.0d0*x2(m)+3.0d0*x3(m)+4.0d0*x4(m)
      x2(m)=x2(m)+3.0d0*x3(m)+6.0d0*x4(m)
      x3(m)=x3(m)+4.0d0*x4(m)

      vx0(m)=vx0(m)+       vx1(m)+       vx2(m)+       vx3(m)+vx4(m)
      vx1(m)=vx1(m)+2.0d0*vx2(m)+3.0d0*vx3(m)+4.0d0*vx4(m)
      vx2(m)=vx2(m)+3.0d0*vx3(m)+6.0d0*vx4(m)
      vx3(m)=vx3(m)+4.0d0*vx4(m)

      yold(m)=y0(m)
      y0(m)=y0(m)+       y1(m)+       y2(m)+       y3(m)+y4(m)
      y1(m)=y1(m)+2.0d0*y2(m)+3.0d0*y3(m)+4.0d0*y4(m)
      y2(m)=y2(m)+3.0d0*y3(m)+6.0d0*y4(m)
      y3(m)=y3(m)+4.0d0*y4(m)

      vy0(m)=vy0(m)+       vy1(m)+       vy2(m)+       vy3(m)+vy4(m)
      vy1(m)=vy1(m)+2.0d0*vy2(m)+3.0d0*vy3(m)+4.0d0*vy4(m)
      vy2(m)=vy2(m)+3.0d0*vy3(m)+6.0d0*vy4(m)
      vy3(m)=vy3(m)+4.0d0*vy4(m)

      zold(m)=z0(m)
      z0(m)=z0(m)+       z1(m)+       z2(m)+       z3(m)+z4(m)
      z1(m)=z1(m)+2.0d0*z2(m)+3.0d0*z3(m)+4.0d0*z4(m)
      z2(m)=z2(m)+3.0d0*z3(m)+6.0d0*z4(m)
      z3(m)=z3(m)+4.0d0*z4(m)

      vz0(m)=vz0(m)+       vz1(m)+       vz2(m)+       vz3(m)+vz4(m)
      vz1(m)=vz1(m)+2.0d0*vz2(m)+3.0d0*vz3(m)+4.0d0*vz4(m)
      vz2(m)=vz2(m)+3.0d0*vz3(m)+6.0d0*vz4(m)
      vz3(m)=vz3(m)+4.0d0*vz4(m)

      vxytemp = vxytemp + vx0(m)*vx0(m)  + vy0(m)*vy0(m)
      vztemp  = vztemp  + vz0(m)*vz0(m)

    ENDDO

    vx2sum  = vx2sum*sfam
    vy2sum  = vy2sum*sfam
    vz2sum  = vz2sum*sfam
    v13sum  = v13sum*sfam
    v23sum  = v23sum*sfam
    vxytemp = vxytemp*sfam
    vztemp  = vztemp*sfam

    vx2sumb = 0.0d0
    vy2sumb = 0.0d0
    vz2sumb = 0.0d0
    v13sumb = 0.0d0
    v23sumb = 0.0d0
    vxytempb= 0.0d0
```

```
vztempb = 0.0d0
DO m=nsfmol+1,nadmol

  vxsq   = vx0(m)*vx0(m)
  vysq   = vy0(m)*vy0(m)
  vzsq   = vz0(m)*vz0(m)
  vx2sumb = vx2sumb+vxsq
  vy2sumb = vy2sumb+vysq
  vz2sumb = vz2sumb+vzsq
  v13sumb = v13sumb+vx0(m)*vz0(m)
  v23sumb = v23sumb+vy0(m)*vz0(m)

  xold(m)=x0(m)
  x0(m)=x0(m)+       x1(m)+       x2(m)+       x3(m)+x4(m)
  x1(m)=x1(m)+2.0d0*x2(m)+3.0d0*x3(m)+4.0d0*x4(m)
  x2(m)=x2(m)+3.0d0*x3(m)+6.0d0*x4(m)
  x3(m)=x3(m)+4.0d0*x4(m)

  vx0(m)=vx0(m)+       vx1(m)+       vx2(m)+       vx3(m)+vx4(m)
  vx1(m)=vx1(m)+2.0d0*vx2(m)+3.0d0*vx3(m)+4.0d0*vx4(m)
  vx2(m)=vx2(m)+3.0d0*vx3(m)+6.0d0*vx4(m)
  vx3(m)=vx3(m)+4.0d0*vx4(m)

  yold(m)=y0(m)
  y0(m)=y0(m)+       y1(m)+       y2(m)+       y3(m)+y4(m)
  y1(m)=y1(m)+2.0d0*y2(m)+3.0d0*y3(m)+4.0d0*y4(m)
  y2(m)=y2(m)+3.0d0*y3(m)+6.0d0*y4(m)
  y3(m)=y3(m)+4.0d0*y4(m)

  vy0(m)=vy0(m)+       vy1(m)+       vy2(m)+       vy3(m)+vy4(m)
  vy1(m)=vy1(m)+2.0d0*vy2(m)+3.0d0*vy3(m)+4.0d0*vy4(m)
  vy2(m)=vy2(m)+3.0d0*vy3(m)+6.0d0*vy4(m)
  vy3(m)=vy3(m)+4.0d0*vy4(m)

  zold(m)=z0(m)
  z0(m)=z0(m)+       z1(m)+       z2(m)+       z3(m)+z4(m)
  z1(m)=z1(m)+2.0d0*z2(m)+3.0d0*z3(m)+4.0d0*z4(m)
  z2(m)=z2(m)+3.0d0*z3(m)+6.0d0*z4(m)
  z3(m)=z3(m)+4.0d0*z4(m)

  vz0(m)=vz0(m)+       vz1(m)+       vz2(m)+       vz3(m)+vz4(m)
  vz1(m)=vz1(m)+2.0d0*vz2(m)+3.0d0*vz3(m)+4.0d0*vz4(m)
  vz2(m)=vz2(m)+3.0d0*vz3(m)+6.0d0*vz4(m)
  vz3(m)=vz3(m)+4.0d0*vz4(m)

  vxytempb = vxytempb + vx0(m)*vx0(m)  + vy0(m)*vy0(m)
  vztempb  = vztempb  + vz0(m)*vz0(m)

ENDDO

vx2sum   = vx2sum+vx2sumb*bfam
vy2sum   = vy2sum+vy2sumb*bfam
vz2sum   = vz2sum+vz2sumb*bfam
vxy2sum  = vx2sum+vy2sum
v2sum    = vxy2sum+vz2sum
v13sum   = v13sum+v13sumb*bfam
v23sum   = v23sum+v23sumb*bfam
```

```
            vxytemp  = vxytemp+vxytempb*bfam
            vztemp   = vztemp+vztempb*bfam

            etua     = v2sum/(3.0d0*nadmol) ! instantaneous T at t
            tmolsum  = tmolsum+etua         ! for average T
            tmol2sum = tmol2sum+etua*etua   ! for T fluctuations
C           ____________________________________________
C
C           Temperature Controls in the parallel (tcl)
C           and perpendicular (tcz) directions
C           ____________________________________________

            tcl0=tcl0+      tcl1+      tcl2+      tcl3+tcl4
            tcl1=tcl1+2.0d0*tcl2+3.0d0*tcl3+4.0d0*tcl4
            tcl2=tcl2+3.0d0*tcl3+6.0d0*tcl4
            tcl3=tcl3+4.0d0*tcl4

            tcz0=tcz0+    tcz1+      tcz2+      tcz3+tcz4
            tcz1=tcz1+2.0d0*tcz2+3.0d0*tcz3+4.0d0*tcz4
            tcz2=tcz2+3.0d0*tcz3+6.0d0*tcz4
            tcz3=tcz3+4.0d0*tcz4
C           ________________________________________
C
C           Adjusting lateral dimensions based on
C           evolution trajectory/history
C           ________________________________________

            bLxold=bLx0
            bLx0=bLx0+      bLx1+      bLx2+      bLx3+bLx4
            bLx1=bLx1+2.0d0*bLx2+3.0d0*bLx3+4.0d0*bLx4
            bLx2=bLx2+3.0d0*bLx3+6.0d0*bLx4
            bLx3=bLx3+4.0d0*bLx4

            bLyold=bLy0
            bLy0=bLy0+       bLy1+      bLy2+      bLy3+bLy4
            bLy1=bLy1+2.0d0*bLy2+3.0d0*bLy3+4.0d0*bLy4
            bLy2=bLy2+3.0d0*bLy3+6.0d0*bLy4
            bLy3=bLy3+4.0d0*bLy4

            IF (istart.eq.2) THEN         ! allowing separation to change
              do i=1,layers
                rsnew(i)       = rsold(i)        - deltz
                rsnew(i+layers)= rsold(i+layers) + deltz
              enddo
            ELSE                          ! fixed separation
              do i=1,layers
                rsnew(i)        = rsold(i)
                rsnew(i+layers) = rsold(i+layers)
              enddo
            ENDIF

            hz0     = rsold(layers+1) - rsold(1)  ! old separation
            hz1     = rsnew(layers+1) - rsnew(1)  ! new separation
            sconold = bLxold*bLyold      ! old confining surface area
            vold    = sconold*hz0        ! old confined volume
            dvold   = 2.0d0*vold
            scon    = bLx0*bLy0          ! new confining surface area
```

```
        dscon   = 2.0d0*scon
c       __________
c
c       FORCE LOOP
c       __________

        CALL accel !!

        totint  = totint+potff  !! fluid-fluid interaction
        totmol  = totmol+potfs  !! fluid-surface interaction

        skztot  = skztot+skz
c       _____________________________
c
c       PRESSURE CONTROL COEFFICIENT
c       _____________________________

        ptemp = (ptemp+vxytemp)
      beta  = 4.0*scon*hz1/(2.0*ptemp-deno)   !! This is k||
        coe   = (-beta*(dscon*pset*hz1-ptemp))/(dscon*hz1*tp)

        pxy    = (pxy+vxy2sum)/dvold   !! parallel P
        pxytot = pxytot+pxy
        pxy2   = pxy2+pxy*pxy

        pz     = (pz+vz2sum)/vold      !! perpendicular P
        pztot  = pztot+pz
        pz2    = pz2+pz*pz

        fsol   = pz-pxy                !! solvation force
        fsoltot= fsoltot+fsol
        fsol2  = fsol2+fsol*fsol

        ppsfsf    = ppsfsf/dvold        ! solvent-solvent to Pxy
        ppsfsftot = ppsfsftot + ppsfsf
        ppsfbf    = ppsfbf/dvold        ! solvent-nanoparticle to Pxy
        ppsfbftot = ppsfbftot + ppsfbf
        ppsfsa    = ppsfsa/dvold        ! solvent-surface to Pxy
        ppsfsatot = ppsfsatot + ppsfsa
        ppbfbf    = ppbfbf/dvold     ! nanoparticle-nanoparticle to Pxy
        ppbfbftot = ppbfbftot + ppbfbf
        ppbfsa    = ppbfsa/dvold        ! nanoparticle-surface to Pxy
        ppbfsatot = ppbfsatot + ppbfsa
        pzsfsf    = pzsfsf/vold
        pzsfsftot = pzsfsftot + pzsfsf
        pzsfbf    = pzsfbf/vold
        pzsfbftot = pzsfbftot + pzsfbf
        pzsfsa    = pzsfsa/vold
        pzsfsatot = pzsfsatot + pzsfsa
        pzbfbf    = pzbfbf/vold
        pzbfbftot = pzbfbftot + pzbfbf
        pzbfsa    = pzbfsa/vold
        pzbfsatot = pzbfsatot + pzbfsa

        fzsf    = 0.50d0*fzsf/sconold  ! perpendicular force/A = Pz
        fzsftot = fzsftot+fzsf
        fzbf    = 0.50d0*fzbf/sconold
```

```
        fzbftot = fzbftot+fzbf

        fxxutot = fxxutot+fxxu/sconold ! parallel force/A = drag
        fxxdtot = fxxdtot+fxxd/sconold

        bLxtot  = bLxtot+bLxold
        bLxt2   = bLxt2+bLxold*bLxold
        bLxold2 = bLxold*bLxold
        bLxt4   = bLxt4+bLxold2*bLxold2
c       _________________________
c
c       GEAR'S CORRECTOR PHASE
c       _________________________

        vcoe = ttnh*(vxytemp/(2.0d0*nadmol*temp)-1.0d0)
        vcoez= ttnh*(vztemp/(nadmol*temp)-1.0d0)

        tclcor = vcoe*h-tcl1
        tcl0   = tcl0+tclcor*f01
        tcl1   = tcl1+tclcor
        tcl2   = tcl2+tclcor*f21
        tcl3   = tcl3+tclcor*f31
        tcl4   = tcl4+tclcor*f41
        fcoe   = ttnh*tcl0

        tczcor = vcoez*h-tcz1
        tcz0   = tcz0+tczcor*f01
        tcz1   = tcz1+tczcor
        tcz2   = tcz2+tczcor*f21
        tcz3   = tcz3+tczcor*f31
        tcz4   = tcz4+tczcor*f41
        fcoez  = ttnh*tcz0

        bLxcor = coe*bLx0*h-bLx1
        bLx0   = bLx0+bLxcor*f01
        bLx1   = bLx1+bLxcor
        bLx2   = bLx2+bLxcor*f21
        bLx3   = bLx3+bLxcor*f31
        bLx4   = bLx4+bLxcor*f41

        schange = schange + bLx0 - bLxold
        IF((schange*schange).ge.drneig) disptest=.true.

      DO m=1,nadmol

          xcor  = (vx0(m)+coe*x0(m))*h-x1(m)
          x0(m) = x0(m)+xcor*f01     !! new positions at (t+dt)
          x1(m) = x1(m)+xcor
          x2(m) = x2(m)+xcor*f21
          x3(m) = x3(m)+xcor*f31
          x4(m) = x4(m)+xcor*f41

          vxcor  = (ax(m)-fcoe*vx0(m))*h-vx1(m)
          vx0(m) = vx0(m)+vxcor*f01  !! new velocities at (t+dt)
          vx1(m) = vx1(m)+vxcor
          vx2(m) = vx2(m)+vxcor*f21
          vx3(m) = vx3(m)+vxcor*f31
```

```
            vx4(m) = vx4(m)+vxcor*f41
            diff(1,m) = diff(1,m)+x0(m)-xold(m)

            ycor  = (vy0(m)+coe*y0(m))*h-y1(m)
            y0(m) = y0(m)+ycor*f01     !! new positions at (t+dt)
            y1(m) = y1(m)+ycor
            y2(m) = y2(m)+ycor*f21
            y3(m) = y3(m)+ycor*f31
            y4(m) = y4(m)+ycor*f41

            vycor  = (ay(m)-fcoe*vy0(m))*h-vy1(m)
            vy0(m) = vy0(m)+vycor*f01  !! new velocities at (t+dt)
            vy1(m) = vy1(m)+vycor
            vy2(m) = vy2(m)+vycor*f21
            vy3(m) = vy3(m)+vycor*f31
            vy4(m) = vy4(m)+vycor*f41
            diff(2,m) = diff(2,m)+y0(m)-yold(m)

            zcor  = vz0(m)*h-z1(m)
            z0(m) = z0(m)+zcor*f01     !! new positions at (t+dt)
            z1(m) = z1(m)+zcor
            z2(m) = z2(m)+zcor*f21
            z3(m) = z3(m)+zcor*f31
            z4(m) = z4(m)+zcor*f41

            vzcor  = (az(m)-fcoez*vz0(m))*h-vz1(m)
            vz0(m) = vz0(m)+vzcor*f01  !! new velocities at (t+dt)
            vz1(m) = vz1(m)+vzcor
            vz2(m) = vz2(m)+vzcor*f21
            vz3(m) = vz3(m)+vzcor*f31
            vz4(m) = vz4(m)+vzcor*f41
            diff(3,m) = diff(3,m)+z0(m)-zold(m)

            dispvec=0.0d0
            DO j=1,3
              dispvec = dispvec+diff(j,m)*diff(j,m)
            ENDDO
            IF(dispvec.ge.drneig)THEN
              disptest =.true.
            ELSE
              disp(1,m) = disp(1,m)+x0(m)-xold(m)
              disp(2,m) = disp(2,m)+y0(m)-yold(m)
            ENDIF
c           ______________________________________
c
c           Apply periodic boundary conditions
c           ______________________________________

            nbcx = dnint(x0(m)/bLx0)
            nbcy = dnint(y0(m)/bLy0)

          if(nbcx.ne.0)then
c             ----------------------------------------
c             Adjustments due to system dimension change
c             ----------------------------------------
              bLx0sq = bLx0*bLx0
              bLx1sq = bLx1*bLx1
```

```
            edotx1 = bLx1/bLx0
            edotx2 = (2.0d0*bLx2*bLx0-bLx1sq)/(bLx0sq)
            edotx3 = ((6.0d0*(bLx3*bLx0-bLx2*bLx1)*bLx0)
     &              +(2.0d0*bLx1*bLx1sq))/(bLx0*bLx0sq)
            edotx4 = ((24.0d0*(bLx4*bLx0sq-bLx3*bLx1*bLx0
     &              +bLx2*bLx1sq)*bLx0)-12.0d0*bLx2*bLx2*bLx0sq
     &              -6.0d0*bLx1sq*bLx1sq)/(bLx0sq*bLx0sq)

            corr0 = -bLx0*nbcx
            corr1 = edotx1*corr0
            corr2 = (edotx1*corr1+edotx2*corr0)/2.0d0
            corr3 = (2.0d0*(edotx1*corr2+edotx2*corr1)
     &             +edotx3*corr0)/6.0d0
            corr4 = (6.0d0*(edotx1*corr3+edotx2*corr2)
     &             +3.0d0*edotx3*corr1+edotx4*corr0)/24.0d0
            x0(m) = x0(m)+corr0
            x1(m) = x1(m)+corr1
            x2(m) = x2(m)+corr2
            x3(m) = x3(m)+corr3
            x4(m) = x4(m)+corr4
          endif

          if(nbcy.ne.0)then
c           ----------------------------------------
c           Adjustments due to system dimension change
c           ----------------------------------------
            bLy0sq = bLy0*bLy0
            bLy1sq = bLy1*bLy1
            edoty1 = bLy1/bLy0
            edoty2 = (2.0d0*bLy2*bLy0-bLy1sq)/(bLy0sq)
            edoty3 = ((6.0d0*(bLy3*bLy0-bLy2*bLy1)*bLy0)
     &              +(2.0d0*bLy1*bLy1sq))/(bLy0*bLy0sq)

            edoty4 = ((24.0d0*(bLy4*bLy0sq-bLy3*bLy1*bLy0
     &              +bLy2*bLy1sq)*bLy0)-12.0d0*bLy2*bLy2*bLy0sq
     &              -6.0d0*bLy1sq*bLy1sq)/(bLy0sq*bLy0sq)
            corr0 = -bLy0*nbcy
            corr1 = edoty1*corr0
            corr2 = (edoty1*corr1+edoty2*corr0)/2.0d0
            corr3 = (2.0d0*(edoty1*corr2+edoty2*corr1)
     &             +edoty3*corr0)/6.0d0
            corr4 = (6.0d0*(edoty1*corr3+edoty2*corr2)
     &             +3.0d0*edoty3*corr1+edoty4*corr0)/24.0d0
            y0(m) = y0(m)+corr0
            y1(m) = y1(m)+corr1
            y2(m) = y2(m)+corr2
            y3(m) = y3(m)+corr3
            y4(m) = y4(m)+corr4
          endif
        ENDDO
c       ___________________________________________
c
c       prepare the surfaces for next iteration
c       ___________________________________________

        do i=1,layers
          rsold(i)        = rsnew(i)
```

```
          rsold(i+layers) = rsnew(i+layers)
        enddo
c       ____________________________________________
c
c       istart=3: equilibration has been reached
c       Record displacement every 'isave' time steps
c       to calculate diffusivity from mean square
c       displacement
c       ____________________________________________

        IF (istart.eq.3) THEN

          icnt = icnt+1
          IF(icnt.eq.isave)then
            icnt  = 0
c           ______________________________
c
c           Calculate density profiles
c           ______________________________

            DO m=1,nsfmol
              ncom=dnint(z0(m)/dz)+ncen
              densf(ncom)=densf(ncom)+1
            ENDDO
            DO j=1,nbfmol
              m = nsfmol+j
              ncom=dnint(z0(m)/dz)+ncen
              denbf(ncom)=denbf(ncom)+1
            ENDDO

            index = index+1
            IF(index.le.itau) THEN

              DO i=1,index-1
                ndt=index-i
                DO m = 1,nsfmol
                  dx = disp(1,m)-rm2(1,m,i)
                  dy = disp(2,m)-rm2(2,m,i)
                  drsf(ndt) = drsf(ndt)+dx*dx+dy*dy
                ENDDO
                DO m = nsfmol+1,nadmol
                  dx = disp(1,m)-rm2(1,m,i)
                  dy = disp(2,m)-rm2(2,m,i)
                  drbf(ndt) = drbf(ndt)+dx*dx+dy*dy
                ENDDO
              ENDDO
              DO m = 1,nadmol
                rm2(1,m,index) = disp(1,m)
                rm2(2,m,index) = disp(2,m)
              ENDDO

            ELSE

              DO m = 1,nsfmol
                dx = disp(1,m)-rm2(1,m,1)
                dy = disp(2,m)-rm2(2,m,1)
                drsf(itau) = drsf(itau)+dx*dx+dy*dy
```

```
            ENDDO
            DO m = nsfmol+1,nadmol
              dx = disp(1,m)-rm2(1,m,1)
              dy = disp(2,m)-rm2(2,m,1)
              drbf(itau) = drbf(itau)+dx*dx+dy*dy
            ENDDO
            DO j = 2,itau
              ndt=(itau+1)-j
              DO m = 1,nsfmol
                dx = disp(1,m)-rm2(1,m,j)
                dy = disp(2,m)-rm2(2,m,j)
                drsf(ndt) = drsf(ndt)+dx*dx+dy*dy
                rm2(1,m,j-1)=rm2(1,m,j)
                rm2(2,m,j-1)=rm2(2,m,j)
              ENDDO
              DO m = nsfmol+1,nadmol
                dx = disp(1,m)-rm2(1,m,j)
                dy = disp(2,m)-rm2(2,m,j)
                drbf(ndt) = drbf(ndt)+dx*dx+dy*dy
                rm2(1,m,j-1)=rm2(1,m,j)
                rm2(2,m,j-1)=rm2(2,m,j)
              ENDDO
            ENDDO
            DO m=1,nadmol
              rm2(1,m,itau)=disp(1,m)
              rm2(2,m,itau)=disp(2,m)
            ENDDO

          ENDIF  !itau

        ENDIF  !isave

      ENDIF
c     _____________________
c
c     INTERMEDIATE RESULTS
c     _____________________

      IF (icntpt.eq.iprint) then

        icntpt  = 0

        tempmol = tmolsum/iprint
        stdtmol = sqrt(tmol2sum/iprint-tempmol*tempmol)*epsi/R
        tempmol = tempmol*epsi/R
        avb     = bLxtot/iprint
        ava     = bLxt2/iprint
        avpxy   = pxytot/iprint
        stdpxy  = sqrt(pxy2/iprint-avpxy*avpxy)
        avpz    = pztot/iprint
        stdpz   = sqrt(pz2/iprint-avpz*avpz)
        avfsol  = fsoltot/iprint
        avfzsf  = fzsftot/iprint
        avfzbf  = fzbftot/iprint
        avfxxd  = fxxdtot/iprint
        avfxxu  = fxxutot/iprint
        avint   = totint/iprint
```

```
            avmol   = totmol/iprint
            avskz   = skztot/iprint

            avppsfsf = ppsfsftot/iprint
            avppsfbf = ppsfbftot/iprint
            avppsfsa = ppsfsatot/iprint
            avppbfbf = ppbfbftot/iprint
            avppbfsa = ppbfsatot/iprint
            avpzsfsf = pzsfsftot/iprint
            avpzsfbf = pzsfbftot/iprint
            avpzsfsa = pzsfsatot/iprint
            avpzbfbf = pzbfbftot/iprint
            avpzbfsa = pzbfsatot/iprint

            tmolsum = 0.0d0
            tmol2sum= 0.0d0
            bLxtot  = 0.0d0
            bLxt2   = 0.0d0
            pxytot  = 0.0d0
            pxy2    = 0.0d0
            pztot   = 0.0d0
            pz2     = 0.0d0
            fsoltot = 0.0d0
            fzsftot = 0.0d0
            fzbftot = 0.0d0
            fxxdtot = 0.0d0
            fxxutot = 0.0d0
            totint  = 0.0d0
            totmol  = 0.0d0
            skztot  = 0.0d0

            ppsfsftot = 0.0d0
            ppsfbftot = 0.0d0
            ppsfsatot = 0.0d0
            ppbfbftot = 0.0d0
            ppbfsatot = 0.0d0
            pzsfsftot = 0.0d0
            pzsfbftot = 0.0d0
            pzsfsatot = 0.0d0
            pzbfbftot = 0.0d0
            pzbfsatot = 0.0d0

c           write(*,*)nstp,hz0,bLx0,pxy,pz,potff,potfs,skz

            write(*,900)nstp,dxtot,tempmol,stdtmol,avb,ava,avint,avmol,
     &      avpxy,stdpxy,avppsfsf,avppsfbf,avppsfsa,avppbfbf,avppbfsa,
     &      avpz,stdpz,avpzsfsf,avpzsfbf,avpzsfsa,avpzbfbf,avpzbfsa,
     &      avfsol,avfzsf,avfzbf,avfxxd,avfxxu,avskz

            IF (istart.eq.2) THEN

              if(nstp.eq.25000)then

c               DO m=1,nadmol
c                 write(2,999)x0(m),y0(m),z0(m),vx0(m),vy0(m),vz0(m)
c               ENDDO
c               write(2,*)bLx0,bLy0,hz1
```

```
              istart = istart + 1

              write(*,*)'_________Shearing Begins__________'
              GO TO 200

            endif

          ELSE

            if (mod(nstp,50000).eq.0) then

              DO m=1,nadmol
                write(2,999)x0(m),y0(m),z0(m),vx0(m),vy0(m),vz0(m)
              ENDDO
              write(2,*)bLx0,bLy0,hz1
              write(2,*)'XXX'
            endif

 900        format(I7,32(1x,E11.5))

          ENDIF

        ENDIF

      ENDDO         !! end of main loop

      DO m=1,nadmol
        write(2,999)x0(m),y0(m),z0(m),vx0(m),vy0(m),vz0(m)
      ENDDO
      write(2,*)bLx0,bLy0,hz1
      write(2,*)nstp,nbfmol,h*sigma/dsqrt(epsi/unitm)

      do i=1,99
        densf(i) = densf(i)/kmax/(dz*ava)
        denbf(i) = denbf(i)/kmax/(dz*ava)
        write(3,999)(i-ncen)*dz,densf(i),denbf(i)
      enddo

      DO i=1,kmax  !! length of record
        DO j=i,min0(kmax,itau+i)
          ndt = j - i
          ncnt1(ndt) = ncnt1(ndt) + 1
        ENDDO
      ENDDO
      drsf(0) = 0.0d0
      drbf(0) = 0.0d0
      do i=0,itau
        drsf(i) = drsf(i)/ncnt1(i)/nsfmol
        if (nbfmol.gt.0) drbf(i) = drbf(i)/ncnt1(i)/nbfmol
      enddo

c     _____________________________________________________
c
c     Fitting least-square line to get limiting slope
c     _____________________________________________________
```

```
      sumx  = 0.0d0
      sumx2 = 0.0d0
      sumxy = 0.0d0
      sumy  = 0.0d0
      sumy2 = 0.0d0
      num   = 3*(itau+1)/4  ! using the last 1/4 section
      DO i=num,itau
        rsqt  = drsf(i)
        xt    = i*isave*h
        sumy  = sumy  + rsqt
        sumy2 = sumy2 + rsqt*rsqt
        sumx  = sumx  + xt
        sumx2 = sumx2 + xt*xt
        sumxy = sumxy + xt*rsqt
      ENDDO
      diff_coef = (sumy*sumx-num*sumxy)
     &            /(sumx*sumx-num*sumx2)/4.0

      write(3,*)'Dsf_rm2=',diff_coef

      sumx  = 0.0d0
      sumx2 = 0.0d0
      sumxy = 0.0d0
      sumy  = 0.0d0
      sumy2 = 0.0d0
      DO i=num,itau
        rsqt  = drbf(i)
        xt    = i*isave*h
        sumy  = sumy  + rsqt
        sumy2 = sumy2 + rsqt*rsqt
        sumx  = sumx  + xt
        sumx2 = sumx2 + xt*xt
        sumxy = sumxy + xt*rsqt
      ENDDO
      diff_coef = (sumy*sumx-num*sumxy)
     &            /(sumx*sumx-num*sumx2)/4.0

      write(3,*)'Dbf_rm2=',diff_coef

      DO i=0,itau,4
        write(3,999)i*isave*h,drsf(i),drbf(i)
      ENDDO

 999  format(20(1x,e11.5))

      stop
      END


      SUBROUTINE setup_sa_position
      IMPLICIT real*8 (a-h,o-z),integer*4 (i-n)
      include 'ljmix'
c     ____________________________________________________
c
c     This SUBROUTINE sets up the surface atoms in a
c     surface unit cell for both confining substrates
c     ____________________________________________________
```

```
      sx(1,1) = 0.0d0
      sy(1,1) = 0.0d0
      sx(2,1) = sx(1,1)-0.50d0*x_L
      sy(2,1) = sy(1,1)-0.50d0*y_L

      sx(1,2) = sx(1,1)
      sy(1,2) = sy(1,1)-y_L/3.0d0
      sx(2,2) = sx(1,1)-0.50d0*x_L
      sy(2,2) = sy(1,1)+y_L/6.0d0

      sx(1,3) = sx(1,1)
      sy(1,3) = sy(1,1)-y_L/3.0d0
      sx(2,3) = sx(1,1)-0.50d0*x_L
      sy(2,3) = sy(1,1)+y_L/6.0d0

      sx(1,4) = sx(1,1)-0.50d0*x_L
      sy(1,4) = sy(1,1)-y_L/6.0d0
      sx(2,4) = sx(1,1)
      sy(2,4) = sy(1,1)+y_L/3.0d0

      sx(1,5) = sx(1,1)-0.50d0*x_L
      sy(1,5) = sy(1,1)-y_L/6.0d0
      sx(2,5) = sx(1,1)
      sy(2,5) = sy(1,1)+y_L/3.0d0
      sx(1,6) = sx(1,1)
      sy(1,6) = sy(1,1)
      sx(2,6) = sx(2,1)
      sy(2,6) = sy(2,1)

      sx(1,7) = sx(1,1)
      sy(1,7) = sy(1,1)
      sx(2,7) = sx(2,1)
      sy(2,7) = sy(2,1)

      sx(1,8) = sx(1,2)
      sy(1,8) = sy(1,2)
      sx(2,8) = sx(2,2)
      sy(2,8) = sy(2,2)

      do i=1,layers
        sx(1,i+layers) = sx(1,i)
        sy(1,i+layers) = sy(1,i)
        sx(2,i+layers) = sx(2,i)
        sy(2,i+layers) = sy(2,i)
      enddo

      RETURN
      END



      SUBROUTINE gauss(vx,vy,vz,tempe)
      IMPLICIT real*8 (a-h,o-z),integer*4 (i-n)
      include 'ljmix'

c     Generate a gaussian random velocity
```

```
      CALL SRAND(ISEED)

      vxpick = 0.0d0
      vypick = 0.0d0
      vzpick = 0.0d0
      DO k=1,12
        ran1   = rand()
        ran2   = rand()
        ran3   = rand()
        vxpick = vxpick+ran1
        vypick = vypick+ran2
        vzpick = vzpick+ran3
      ENDDO
      vx = (vxpick-6.0d0)*dsqrt(tempe)
      vy = (vypick-6.0d0)*dsqrt(tempe)
      vz = (vzpick-6.0d0)*dsqrt(tempe)

      RETURN
      END



      SUBROUTINE fluid_neighbor_list
      IMPLICIT real*8 (a-h,o-z),integer*4 (i-n)
      include 'ljmix'
c     ____________________________________________________________
c
c     Construct neighbor lists of other UA's for molecules
c     ____________________________________________________________

C     ------------------------
C     Like pair of smaller-smaller fluid atoms
C     ------------------------

      DO m=1,nsfmol+1
        inblss(m) = 0
        inblsb(m) = 0
      ENDDO
      DO m=1,nnblss
        inblstss(m) = 0
      ENDDO
      DO m=1,nnblsb
        inblstsb(m) = 0
      ENDDO

      DO m=1,nbfmol+1
        inblbb(m) = 0
      ENDDO
      DO m=1,nnblbb
        inblstbb(m) = 0
      ENDDO

      nlistss = 0
      nlistsb = 0
      DO m=1,nsfmol
```

```
C       ---------------------
C       solvent-solvent pairs
C       ---------------------
        inblss(m) = nlistss+1
        DO i=m+1,nsfmol
          xij  = x0(m)-x0(i)
          yij  = y0(m)-y0(i)
          zij  = z0(m)-z0(i)
          xij = xij-bLx0*dnint(xij/bLx0)
          yij = yij-bLy0*dnint(yij/bLy0)
          rij2=xij*xij+yij*yij+zij*zij
          IF (rij2.le.rnbl2sfsf) then
            nlistss=nlistss+1
            inblstss(nlistss)=i    !! molecule
          ENDIF
        ENDDO

C       -------------------------
C       solvent-nanoparticle pairs
C       -------------------------
        inblsb(m) = nlistsb+1
        DO i=nsfmol+1,nadmol
          xij  = x0(m)-x0(i)
          yij  = y0(m)-y0(i)
          zij  = z0(m)-z0(i)
          xij = xij-bLx0*dnint(xij/bLx0)
          yij = yij-bLy0*dnint(yij/bLy0)
          rij2=xij*xij+yij*yij+zij*zij
          IF (rij2.le.rnbl2sfbf) then
            nlistsb=nlistsb+1
            inblstsb(nlistsb)=i    !! molecule
          ENDIF

        ENDDO

      ENDDO

      nlistbb = 0
      DO j=1,nbfmol
        m = j + nsfmol
C       ------------------------------
C       nanoparticle-nanoparticle pairs
C       ------------------------------
        inblbb(j) = nlistbb+1
        DO i=m+1,nadmol
          xij  = x0(m)-x0(i)
          yij  = y0(m)-y0(i)
          zij  = z0(m)-z0(i)
          xij = xij-bLx0*dnint(xij/bLx0)
          yij = yij-bLy0*dnint(yij/bLy0)
          rij2=xij*xij+yij*yij+zij*zij
          IF (rij2.le.rnbl2bfbf) then
            nlistbb=nlistbb+1
            inblstbb(nlistbb)=i    !! molecule
          ENDIF
        ENDDO
```

```
ENDDO

IF(nlistss.gt.nnblss) then
  print *,'make nnblss > ',nlistss+1
  stop
ENDIF
IF(nlistsb.gt.nnblsb) then
  print *,'make nnblsb > ',nlistsb+1
  stop
ENDIF
IF(nlistbb.gt.nnblbb) then
  print *,'make nnblbb > ',nlistbb+1
  stop
ENDIF

RETURN
END



SUBROUTINE accel
IMPLICIT real*8 (a-h,o-z),integer*4 (i-n)
include 'ljmix'

DO m=1,nadmol
  ax(m) = 0.0d0
  ay(m) = 0.0d0
  az(m) = 0.0d0
ENDDO

ppsfsf = 0.0d0
ppsfbf = 0.0d0
ppsfsa = 0.0d0
ppbfbf = 0.0d0
ppbfsa = 0.0d0
pzsfsf = 0.0d0
pzsfbf = 0.0d0
pzsfsa = 0.0d0
pzbfbf = 0.0d0
pzbfsa = 0.0d0

fzsf = 0.0d0
fzbf = 0.0d0
fxxd = 0.0d0
fxxu = 0.0d0

potsfsf = 0.0d0
potbfbf = 0.0d0
potsfsa = 0.0d0
potbfsa = 0.0d0
ptemp = 0.0d0
deno  = 0.0d0

skzc  = 0.0d0
skzs  = 0.0d0
```

```
C     ------------------------------------------------------------
C     dxtot: displacement of the upper surface
C            for continuous shearing (stationary lower surface)
C     ------------------------------------------------------------
      dxtot = dxtot + deltx
C     _______________________________
C
C     INTERACTIONS BASED ON SOLVENT
C     _______________________________

      DO m=1,nsfmol

        xorig = xold(m)
        yorig = yold(m)
        zorig = zold(m)

        rkz  = (zorig-rsold(1))*zk
        skzc = skzc+cos(rkz)
        skzs = skzs+sin(rkz)

        xpred = x0(m)
        ypred = y0(m)
        zpred = z0(m)
        aclx  = ax(m)
        acly  = ay(m)
        aclz  = az(m)
C       ____________________________________
C
C       solvent-solvent pair interactions
C       ____________________________________

        ibegss = inblss(m)
        iendss = inblss(m+1)-1

        DO ilist=ibegss,iendss   ! No. of interacting pairs
          i=inblstss(ilist)      ! identity of interacting partner
C         ------------------------------------------
C         Use old coordinates to calculate properties
C         ------------------------------------------
          Xij  = xorig - xold(i)  !! old positions
          Yij  = yorig - yold(i)
          Zij  = zorig - zold(i)
          xpbc = -bLxold*dnint(Xij/bLxold)
          ypbc = -bLyold*dnint(Yij/bLyold)
          Xij  = Xij+xpbc
          Yij  = Yij+ypbc
          xij2 = Xij*Xij
          yij2 = Yij*Yij
          rxy2 = xij2 + yij2
          zij2 = Zij*Zij
          rij2 = rxy2 + zij2
          IF (rij2.le.rcut2sfsf) then
            rho2  = sig2sfsf/rij2
            rho6  = rho2*rho2*rho2
            rho12 = rho6*rho6
            rij   = dsqrt(rij2)
            Fij   = eps24sfsf*((2.0d0*rho12-rho6)/rij2 -fcsfsf/rij)
```

```
            ppsfsf = ppsfsf + Fij*rxy2
            pzsfsf = pzsfsf + Fij*zij2

            potsfsf = potsfsf+(rho12-rho6)-vcsfsf+fcvsfsf*rij
          ENDIF
C         ------------------------------------------------
C         Use predicted coordinates to estimate properties
C         ------------------------------------------------
          x0ij  = xpred - x0(i)  !! predicted positions for calculating
          y0ij  = ypred - y0(i)  !! accelerations and coefficients
          z0ij  = zpred - z0(i)
          x0ij  = x0ij + xpbc
          y0ij  = y0ij + ypbc
          r0xy2 = x0ij*x0ij + y0ij*y0ij
          z0ij2 = z0ij*z0ij
          r0ij2 = r0xy2 + z0ij2
          IF (r0ij2.le.rcut2sfsf) then
          r0ij4 = r0ij2*r0ij2
          r0xy4 = r0xy2*r0xy2
            rho2  = sig2sfsf/r0ij2
            rho6  = rho2*rho2*rho2
            rho12 = rho6*rho6
            r0ij  = dsqrt(r0ij2)
            F0ij  = eps24sfsf*((2.0d0*rho12-rho6)-fcsfsf*r0ij)/r0ij2
            x0ff_r4 = epssfsf*(-672.0d0*rho12+192.0d0*rho6
     &               +24.0d0*r0ij*fcsfsf)/r0ij4

            ptemp = ptemp+F0ij*r0xy2
            deno  = deno+r0xy4*x0ff_r4+2.0*F0ij*r0xy2

            fx0lj = F0ij*(X0ij) !! Fij=(-dv/dr)/r
            fy0lj = F0ij*(Y0ij)
            fz0lj = F0ij*(Z0ij)
            aclx  = aclx + fx0lj
            acly  = acly + fy0lj
            aclz  = aclz + fz0lj
            ax(i) = ax(i) - fx0lj
            ay(i) = ay(i) - fy0lj
            az(i) = az(i) - fz0lj
          ENDIF

          ENDDO
C         ___________________________________________
C
C         solvent-nanoparticle pair interactions
C         ___________________________________________

          ibegsb = inblsb(m)
          iendsb = inblsb(m+1)-1

          DO ilist=ibegsb,iendsb  ! No. of interacting nanoparticles
            j=inblstsb(ilist)     ! identity of interacting nanoparticles
C           ----------------------------------------
C           Use old coordinates to calculate properties
C           ----------------------------------------
            Xij  = xorig - xold(j)  !! old positions
```

```
            Yij  = yorig - yold(j)
            Zij  = zorig - zold(j)
            xpbc = -bLxold*dnint(Xij/bLxold)
            ypbc = -bLyold*dnint(Yij/bLyold)
            Xij  = Xij+xpbc
            Yij  = Yij+ypbc

            xij2 = Xij*Xij
            yij2 = Yij*Yij
            rxy2 = xij2 + yij2
            zij2 = Zij*Zij
            rij2 = rxy2 + zij2
            IF (rij2.le.rcut2sfbf) then
              rho2  = sig2sfbf/rij2
              rho6  = rho2*rho2*rho2
              rho12 = rho6*rho6
              rij   = dsqrt(rij2)
              Fij   = eps24sfbf*((2.0d0*rho12-rho6)/rij2 -fcsfbf/rij)

              ppsfbf = ppsfbf + Fij*rxy2
              pzsfbf = pzsfbf + Fij*zij2

              potsfbf = potsfbf+(rho12-rho6)-vcsfbf+fcvsfbf*rij
            ENDIF
C           ------------------------------------------------
C           Use predicted coordinates to estimate properties
C           ------------------------------------------------
            x0ij  = xpred - x0(j)  !! predicted positions for calculating
            y0ij  = ypred - y0(j)  !! accelerations and coefficients
            z0ij  = zpred - z0(j)
            x0ij  = x0ij + xpbc
            y0ij  = y0ij + ypbc
            r0xy2 = x0ij*x0ij + y0ij*y0ij
            z0ij2 = z0ij*z0ij
            r0ij2 = r0xy2 + z0ij2
            IF (r0ij2.le.rcut2sfbf) then
            r0ij4 = r0ij2*r0ij2
            r0xy4 = r0xy2*r0xy2
              rho2  = sig2sfbf/r0ij2
              rho6  = rho2*rho2*rho2
              rho12 = rho6*rho6
              r0ij  = dsqrt(r0ij2)
              F0ij  = eps24sfbf*((2.0d0*rho12-rho6)-fcsfbf*r0ij)/r0ij2
              x0ff_r4 = epssfbf*(-672.0d0*rho12+192.0d0*rho6
     &                +24.0d0*r0ij*fcsfsf)/r0ij4

              ptemp = ptemp+F0ij*r0xy2
              deno  = deno+r0xy4*x0ff_r4+2.0*F0ij*r0xy2

              fx0lj = F0ij*(X0ij) !! Fij=(-dv/dr)/r
              fy0lj = F0ij*(Y0ij)
              fz0lj = F0ij*(Z0ij)
              aclx  = aclx + fx0lj
              acly  = acly + fy0lj
              aclz  = aclz + fz0lj
              ax(j) = ax(j) - fx0lj
              ay(j) = ay(j) - fy0lj
```

```
            az(j) = az(j) - fz0lj
          ENDIF

          ENDDO
C         _______________________________
C
C         solvent-surface interactions
C         _______________________________

C         --------------------------
C         stationary (lower) surface
C         --------------------------
          nyo    = dnint(yorig/y_L)
          yshfto = yorig - nyo*y_L
          nyp    = dnint(ypred/y_L)
          yshftp = ypred - nyp*y_L

          nxo    = dnint(xorig/x_L)
          xshfto = xorig - nxo*x_L
          nxp    = dnint(xpred/x_L)
          xshftp = xpred - nxp*x_L

          DO k=1,layers           !stationary surface
            zij  = zorig - rsold(k)
            zij2 = zij*zij
            IF (zij2.le.rcut2sfsa) THEN

              DO k1=1,2
                DO i=-3,3
                  syad = i*y_L
                  rsy  = sy(k1,k) + syad
                  DO j=-4,4
                    sxad = j*x_L
                    rsx  = sx(k1,k) + sxad
                    Xij  = xshfto - rsx
                    Yij  = yshfto - rsy
                    xij2 = Xij*Xij
                    yij2 = Yij*Yij
                    rxy2 = xij2 + yij2
                    rij2 = rxy2 + zij2
                    IF (rij2.le.rcut2sfsa) then
                      rij  = dsqrt(rij2)
                      rho2 = sig2sfsa/rij2
                      rho6 = rho2*rho2*rho2
                      rho12= rho6*rho6
                     Fij = eps24sfsa*((2.0d0*rho12-rho6)-fcsfsa*rij)/rij2
                      potsfsa = potsfsa+((rho12-rho6)-vcsfsa+fcvsfsa*rij)
                      fzsf   = fzsf + Fij*abs(zij)
                      ppsfsa = ppsfsa + Fij*rxy2
                      fxxd   = fxxd + Fij*Xij
                      dis    = zorig + (hz0/2.0)
                      pzsfsa = pzsfsa + Fij*zij*dis
                    ENDIF

                    Xij = xshftp - rsx
                    Yij = yshftp - rsy
                    Zij = zpred - rsnew(k)
```

```
                  rxy2 = Xij*Xij + Yij*Yij
                  zij2 = Zij*Zij
                  rij2 = rxy2 + zij2
                  IF (rij2.le.rcut2sfsa) then
                    rxy4 = rxy2*rxy2
                    rij4 = rij2*rij2
                    rho2 = sig2sfsa/rij2
                    rho6 = rho2*rho2*rho2
                    rho12= rho6*rho6
                    rij  = dsqrt(rij2)
                   Fij = eps24sfsa*((2.0d0*rho12-rho6)/rij2-fcsfsa/rij)
                    xwf_r4 = epssfsa*(-672.0d0*rho12+192.0d0*rho6
     &                     +24.0d0*rij*fcsfsa)/rij4
                    rxys  = Xij*xpred+Yij*ypred
                  ptemp = ptemp+Fij*rxy2
                    deno  = deno+rxy2*rxys*xwf_r4ws+2.0*Fij*rxys

                    fxlj = Fij*(Xij) !! Fij=(-dv/dr)/r
                    fylj = Fij*(Yij)
                    fzlj = Fij*(Zij)
                    aclx = aclx + fxlj
                    acly = acly + fylj
                    aclz = aclz + fzlj
                  ENDIF

                enddo  ! do nx
              enddo    ! do ny
            enddo      ! do k1

          endif
        enddo !!k

C       -------------------------------
C       moving/shearing (upper) surface
C       -------------------------------
        xshfto = xorig - dxtot
        nxo    = dnint(xshfto/x_L)
        xshfto = xshfto - nxo*x_L
        xshftp = xpred - dxtot
        nxp    = dnint(xshftp/x_L)
        xshftp = xshftp - nxp*x_L

        DO k=layers+1,2*layers   ! moving/searing surface
          zij  = zorig - rsold(k)
          zij2 = zij*zij
          IF (zij2.le.rcut2sfsa) THEN

            DO k1=1,2
              DO i=-3,3
                syad = i*y_L
                rsy  = sy(k1,k) + syad
                DO j=-4,4
                  sxad = j*x_L
                  rsx  = sx(k1,k) + sxad
                  Xij  = xshfto - rsx
                  Yij  = yshfto - rsy
                  xij2 = Xij*Xij
```

```
                yij2 = Yij*Yij
                rxy2 = xij2 + yij2
                rij2 = rxy2 + zij2
                IF (rij2.le.rcut2sfsa) then
                  rij  = dsqrt(rij2)
                  rho2 = sig2sfsa/rij2
                  rho6 = rho2*rho2*rho2
                  rho12= rho6*rho6
                 Fij = eps24sfsa*((2.0d0*rho12-rho6)-fcsfsa*rij)/rij2
                  potsfsa = potsfsa+((rho12-rho6)-vcsfsa+fcvsfsa*rij)
                  fzsf   = fzsf + Fij*abs(zij)
                  ppsfsa = ppsfsa + Fij*rxy2
                  fxxu   = fxxu + Fij*Xij
                  dis    = zorig - (hz0/2.0)
                  pzsfsa = pzsfsa + Fij*zij*dis
                ENDIF

                Xij = xshftp - rsx
                Yij = yshftp - rsy
                Zij = zpred - rsnew(k)
                rxy2 = Xij*Xij + Yij*Yij
                zij2 = Zij*Zij
                rij2 = rxy2 + zij2
                IF (rij2.le.rcut2sfsa) then
                  rxy4 = rxy2*rxy2
                  rij4 = rij2*rij2
                  rho2 = sig2sfsa/rij2
                  rho6 = rho2*rho2*rho2
                  rho12= rho6*rho6
                  rij  = dsqrt(rij2)
                 Fij = eps24sfsa*((2.0d0*rho12-rho6)/rij2-fcsfsa/rij)
                  xwf_r4 = epssfsa*(-672.0d0*rho12+192.0d0*rho6
&                     +24.0d0*rij*fcsfsa)/rij4!! added truncated term
                  rxys  = Xij*xpred+Yij*ypred
                  ptemp = ptemp+Fij*rxy2
                  deno  = deno+rxy2*rxys*xwf_r4ws+2.0*Fij*rxys

                  fxlj = Fij*(Xij) !! Fij=(-dv/dr)/r
                  fylj = Fij*(Yij)
                  fzlj = Fij*(Zij)
                  aclx = aclx + fxlj
                  acly = acly + fylj
                  aclz = aclz + fzlj
                ENDIF

              enddo  ! do nx
            enddo    ! do ny
          enddo      ! do k1

        endif
      enddo !!k

      ax(m) = aclx/sfam
      ay(m) = acly/sfam
      az(m) = aclz/sfam

     ENDDO
```

```
C     ____________________________________
C
C     INTERACTIONS BASED ON NANOPARTICLE
C     ____________________________________

      DO jj=1,nbfmol
        m = jj+nsfmol

        xorig = xold(m)
        yorig = yold(m)
        zorig = zold(m)

        xpred = x0(m)
        ypred = y0(m)
        zpred = z0(m)
        aclx  = ax(m)
        acly  = ay(m)
        aclz  = az(m)
C       ________________________________________________
C
C       nanoparticle-nanoparticle pair interactions
C       ________________________________________________

        ibegbb = inblbb(jj)
        iendbb = inblbb(jj+1)-1

        DO ilist=ibegbb,iendbb
          k=inblstbb(ilist)
C         ------------------------------------------
C         Use old coordinates to calculate properties
C         ------------------------------------------
          Xij  = xorig - xold(k)  !! old positions
          Yij  = yorig - yold(k)
          Zij  = zorig - zold(k)
          xpbc = -bLxold*dnint(Xij/bLxold)
          ypbc = -bLyold*dnint(Yij/bLyold)
          Xij  = Xij+xpbc
          Yij  = Yij+ypbc
          xij2 = Xij*Xij
          yij2 = Yij*Yij
          rxy2 = xij2 + yij2
          zij2 = Zij*Zij
          rij2 = rxy2 + zij2
          IF (rij2.le.rcut2bfbf) then
            rho2  = sig2bfbf/rij2
            rho6  = rho2*rho2*rho2
            rho12 = rho6*rho6
            rij   = dsqrt(rij2)
            Fij   = eps24bfbf*((2.0d0*rho12-rho6)/rij2 -fcbfbf/rij)

            ppbfbf = ppbfbf + Fij*rxy2
            pzbfbf = pzbfbf + Fij*zij2

            potbfbf = potbfbf+(rho12-rho6)-vcbfbf+fcvbfbf*rij
          ENDIF
```

```
C         ------------------------------------------------
C         Use predicted coordinates to estimate properties
C         ------------------------------------------------
          x0ij  = xpred - x0(k)  !! predicted positions for calculating
          y0ij  = ypred - y0(k)  !! accelerations and coefficients
          z0ij  = zpred - z0(k)
          x0ij  = x0ij-bLx0*dnint(x0ij/bLx0)
          y0ij  = y0ij-bLy0*dnint(y0ij/bLy0)
          r0xy2 = x0ij*x0ij + y0ij*y0ij
          z0ij2 = z0ij*z0ij
          r0ij2 = r0xy2 + z0ij2
          IF (r0ij2.le.rcut2bfbf) then
          r0ij4 = r0ij2*r0ij2
          r0xy4 = r0xy2*r0xy2
            rho2  = sig2bfbf/r0ij2
            rho6  = rho2*rho2*rho2
            rho12 = rho6*rho6
            r0ij  = dsqrt(r0ij2)
            F0ij  = eps24bfbf*((2.0d0*rho12-rho6)-fcbfbf*r0ij)/r0ij2
            x0ff_r4=epsbfbf*(-672.0d0*rho12+192.0d0*rho6
     &              +24.0d0*r0ij*fcbfbf)/r0ij4  !! added truncated term

            ptemp = ptemp+F0ij*r0xy2
            deno  = deno+r0xy4*x0ff_r4+2.0*F0ij*r0xy2

            fx0lj = F0ij*(X0ij) !! Fij=(-dv/dr)/r
            fy0lj = F0ij*(Y0ij)
            fz0lj = F0ij*(Z0ij)
            aclx  = aclx + fx0lj
            acly  = acly + fy0lj
            aclz  = aclz + fz0lj
            ax(k) = ax(k) - fx0lj
            ay(k) = ay(k) - fy0lj
            az(k) = az(k) - fz0lj
          ENDIF

          ENDDO
C         ___________________________________________
C
C         nanoparticle-surface pair interactions
C         ___________________________________________

C         --------------------------
C         stationary (lower) surface
C         --------------------------
          nyo    = dnint(yorig/y_L)
          yshfto = yorig - nyo*y_L
          nyp    = dnint(ypred/y_L)
          yshftp = ypred - nyp*y_L

          nxo    = dnint(xorig/x_L)
          xshfto = xorig - nxo*x_L
          nxp    = dnint(xpred/x_L)
          xshftp = xpred - nxp*x_L
          DO k=1,layers
            zij  = zorig - rsold(k)
            zij2 = zij*zij
```

```
      IF (zij2.le.rcut2bfsa) THEN

        DO k1=1,2
          DO i=-4,4
            syad = i*y_L
            rsy  = sy(k1,k) + syad
            DO j=-6,6
              sxad = j*x_L
              rsx  = sx(k1,k) + sxad
              Xij  = xshfto - rsx
              Yij  = yshfto - rsy
              xij2 = Xij*Xij
              yij2 = Yij*Yij
              rxy2 = xij2 + yij2
              rij2 = rxy2 + zij2
              IF (rij2.le.rcut2bfsa) then
                rij  = dsqrt(rij2)
                rho2 = sig2bfsa/rij2
                rho6 = rho2*rho2*rho2
                rho12= rho6*rho6
               Fij = eps24bfsa*((2.0d0*rho12-rho6)-fcbfsa*rij)/rij2

                potbfsa = potbfsa+((rho12-rho6)-vcbfsa+fcvbfsa*rij)
                fzbf = fzbf + Fij*abs(zij)
                ppbfsa = ppbfsa + Fij*rxy2
                fxxd   = fxxd   + Fij*Xij
                dis    = zorig  +(hz0/2.0)
                pzbfsa = pzbfsa + Fij*zij*dis
              ENDIF

              Xij = xshftp - rsx
              Yij = yshftp - rsy
              Zij = zpred  - rsnew(k)
              rxy2 = Xij*Xij + Yij*Yij
              zij2 = Zij*Zij
              rij2 = rxy2 + zij2
              IF (rij2.le.rcut2bfsa) then
                rxy4 = rxy2*rxy2
                rij4 = rij2*rij2
                rho2 = sig2bfsa/rij2
                rho6 = rho2*rho2*rho2
                rho12= rho6*rho6
                rij  = dsqrt(rij2)
               Fij = eps24bfsa*((2.0d0*rho12-rho6)/rij2-fcbfsa/rij)
                xwf_r4 = epsbfsa*(-672.0d0*rho12+192.0d0*rho6
     &            +24.0d0*rij*fcbfsa)/rij4  !! added truncated term
                rxys  = Xij*xpred+Yij*ypred
                ptemp = ptemp+Fij*rxy2
                deno  = deno+rxy2*rxys*xwf_r4+2.0*Fij*rxys

                fxlj = Fij*(Xij) !! Fij=(-dv/dr)/r
                fylj = Fij*(Yij)
                fzlj = Fij*(Zij)
                aclx = aclx + fxlj
                acly = acly + fylj
                aclz = aclz + fzlj
              ENDIF
```

```
                enddo  ! do nx
              enddo    ! do ny
            enddo      ! do k1

          endif
        enddo !!kk

C       -------------------------------
C       moving/shearing (upper) surface
C       -------------------------------
        xshfto = xorig - dxtot
        nxo    = dnint(xshfto/x_L)
        xshfto = xshfto - nxo*x_L
        xshftp = xpred - dxtot
        nxp    = dnint(xshftp/x_L)
        xshftp = xshftp - nxp*x_L
        DO k=layers+1,2*layers !moving
          zij  = zorig - rsold(k)
          zij2 = zij*zij
          IF (zij2.le.rcut2bfsa) THEN

            DO k1=1,2
              DO i=-4,4
                syad = i*y_L
                rsy  = sy(k1,k) + syad
                DO j=-6,6
                  sxad = j*x_L
                  rsx  = sx(k1,k) + sxad
                  Xij  = xshfto - rsx
                  Yij  = yshfto - rsy
                  xij2 = Xij*Xij
                  yij2 = Yij*Yij
                  rxy2 = xij2 + yij2
                  rij2 = rxy2 + zij2
                  IF (rij2.le.rcut2bfsa) then
                    rij  = dsqrt(rij2)
                    rho2 = sig2bfsa/rij2
                    rho6 = rho2*rho2*rho2
                    rho12= rho6*rho6
                   Fij = eps24bfsa*((2.0d0*rho12-rho6)-fcbfsa*rij)/rij2
                    potbfsa = potbfsa+((rho12-rho6)-vcbfsa+fcvbfsa*rij)
                    fzbf = fzbf + Fij*abs(zij)
                    ppbfsa = ppbfsa + Fij*rxy2
                    fxxu   = fxxu + Fij*Xij
                    dis    = zorig - (hz0/2.0)
                    pzbfsa = pzbfsa + Fij*zij*dis
                  ENDIF

                  Xij = xshftp - rsx
                  Yij = yshftp - rsy
                  Zij = zpred  - rsnew(k)
                  rxy2 = Xij*Xij + Yij*Yij
                  zij2 = Zij*Zij
                  rij2 = rxy2 + zij2
                  IF (rij2.le.rcut2bfsa) then
                    rxy4 = rxy2*rxy2
```

```
                rij4 = rij2*rij2
                rho2 = sig2bfsa/rij2
                rho6 = rho2*rho2*rho2
                rho12= rho6*rho6
                rij  = dsqrt(rij2)
               Fij = eps24bfsa*((2.0d0*rho12-rho6)/rij2-fcbfsa/rij)

                xwf_r4 = epsbfsa*(-672.0d0*rho12+192.0d0*rho6
&                 +24.0d0*rij*fcbfsa)/rij4  !! added truncated term
                rxys  = Xij*xpred+Yij*ypred
                ptemp = ptemp+Fij*rxy2
                deno  = deno+rxy2*rxys*xwf_r4+2.0*Fij*rxys

                fxlj = Fij*(Xij) !! Fij=(-dv/dr)/r
                fylj = Fij*(Yij)
                fzlj = Fij*(Zij)
                aclx = aclx + fxlj
                acly = acly + fylj
                aclz = aclz + fzlj
              ENDIF

            enddo  ! do nx
          enddo    ! do ny
        enddo      ! do k1

      endif

    enddo !!k

    ax(m) = aclx/bfam
    ay(m) = acly/bfam
    az(m) = aclz/bfam

  ENDDO

  pxy   = ppsfsf + ppsfbf + ppbfbf + ppsfsa + ppbfsa
  pz    = pzsfsf + pzsfbf + pzbfbf + pzsfsa + pzbfsa
  potff = eps4sfsf*potsfsf+eps4sfbf*potsfbf+eps4bfbf*potbfbf
  potfs = eps4sfsa*potsfsa+eps4bfsa*potbfsa

  skzc = skzc/nsfmol
  skzs = skzs/nsfmol
  skz  = dsqrt(skzc*skzc+skzs*skzs)

  RETURN
  END
```

```
 PARAMETER (layers=8)             ! no. of layers per surface
 PARAMETER (nbfmol=5)             ! no. of nanoparticles
 PARAMETER (nadmol=1600)          ! total no. of particles
 PARAMETER (nsfmol=nadmol-nbfmol)
 PARAMETER (nnblss=nsfmol*150)
 PARAMETER (nnblsb=nsfmol*10)
 PARAMETER (nnblbb=nbfmol*6+200)
 PARAMETER (tset=300)             ! system temperature in K
 PARAMETER (ppara=1.00)           ! parallel pressure in atm.
 PARAMETER (unorm=0.00)           ! compressing velocity m/s
 PARAMETER (upara=0.00)           ! shearing velocity m/s
 PARAMETER (nsimut=400000)        ! total no. of time steps
 PARAMETER (R=8.314d0)            ! gas constant in SI unit
 PARAMETER (iseed=45)             ! seed number
 PARAMETER (nbin=99)              ! no. of bins
 PARAMETER (iprint=1000)
 PARAMETER (isave=25)
 PARAMETER (kmax=nsimut/isave)
 PARAMETER (itau=4095)

 COMMON /systemvar/h,temp,al,x_L,y_L,z_L,boxx,boxy,pi,
&       scon,sconold,bLx0,bLy0,hz0,dhz0,hz,hz0old,xk,yk,zk,
&       sk,skdb,skz,bLxold,bLyold,sk1,sfam,bfam,drneig,
&       pconv,uconv,pset,deltx,deltz,dxtot

 COMMON /intvar/nstp,ncnt1(0:itau)

 COMMON /potenvar/epsi,sigma,unitm,epssfsf,eps4sfsf,eps24sfsf,
&       sigsfsf,sig2sfsf,rcut2sfsf,rnbl2sfsf,vcsfsf,fcsfsf,
&       fcvsfsf,epssfbf,eps4sfbf,eps24sfbf,sigsfbf,sig2sfbf,
&       rcut2sfbf,rnbl2sfbf,vcsfbf,fcsfbf,fcvsfbf,epssfsa,
&       eps4sfsa,eps24sfsa,sigsfsa,sig2sfsa,rcut2sfsa,rnbl2sfsa,
&       vcsfsa,fcsfsa,fcvsfsa,epsbfbf,eps4bfbf,eps24bfbf,
&       sigbfbf,sig2bfbf,rcut2bfbf,rnbl2bfbf,vcbfbf,fcbfbf,
&       fcvbfbf,epsbfsa,eps4bfsa,eps24bfsa,sigbfsa,sig2bfsa,
&       rcut2bfsa,rnbl2bfsa,vcbfsa,fcbfsa,fcvbfsa

 COMMON /gearvar/h0,hold,ptemp,beta,coetop,coebot,deno

 COMMON /particle/x0(nadmol),y0(nadmol),z0(nadmol),
&       xold(nadmol),yold(nadmol),zold(nadmol),
&       vx0(nadmol),vy0(nadmol),vz0(nadmol),
&       ax(nadmol),ay(nadmol),az(nadmol)

 COMMON /surf/rsold(2*layers),rsnew(2*layers),
&       sx(2,2*layers),sy(2,2*layers)

 COMMON /adadneig/inblss(nsfmol+1),inblstss(nnblss),
&       inblsb(nsfmol+1),inblstsb(nnblsb),
&       inblbb(nbfmol+1),inblstbb(nnblbb)

 COMMON /property/ppsfsf,ppsfbf,ppbfbf,ppsfsa,ppbfsa,pzsfsf,
&       pzsfbf,pzbfbf,pzsfsa,pzbfsa,pxy,pz,fzsf,fzbf,fxxu,fxxd,
&       potff,potfs,potsfsf,potsfbf,potbfbf,potsfsa,potbfsa

 COMMON /transport/drsf(0:itau),drbf(0:itau),rm2(2,nadmol,itau),
&       dr(0:itau)
```

BIBLOGRAPHY

[1] K. Eric Drexler, *Engines of Creation*: *The Coming Era of Nanotechnology*, Anchor books, New York (1986).

[2] Richard Feynman, *Eng. Sci.* **23**, 22 (1960).

[3] H. Yoshizawa and J. N. Israelachivili, *J. Phys. Chem.* **75**, 1400 (1981).

[4] B. Bhushan (ed.), *Handbook of Micro/Nanotribology*, CRC Press, Boca Raton, FL, (1998).

[5] R. G. Horn and J. N. Israelachvili, *J. Chem. Phys.* **75**, 1400 (1981).

[6] H. K. Christenson, *J. Chem. Phys.* **78**, 6906 (1983).

[7] M. L. Gee, P. M. McGuiggan, J. N. Israelachvili, and A. M. Homola, *J. Chem. Phys*. **93**, 1895 (1990).

[8] A. L. Demirel and S. Granick, *Phy. Rev. Lett.* **77**, 2261 (1996).

[9] J. Klein and E. Kumacheva, *J. Chem. Phys.* **108**, 6996 (1998).

[10] C. M. Mate, G. C. McClelland, R. Erlandsson, and S. Chiang, *Phy. Rev. Lett.* **59**, 1942 (1987).

[11] R. Erlandsson, G. Hadziioannou, C. M. Mate, G. C. McClelland, and S. Chiang, *J. Chem. Phys.* **89**, 5190 (1988).

[12] C. M. Mate, *Phy. Rev. Lett.* **68**, 3323 (1992).

[13] R. Overney, E. Meyer, J. Frommer, D. Brodbeck, R. Luthi, L. Howald, H.-J. Guntherodt, M. Fujihira, H. Takano, and Y. Gotoh, *Nature* **359**, 133 (1992).

[14] R. Overney and E. Meyer, *MRS Bulletin* **18**, 26 (1993).

[15] J. Krim and A. Wisdom, *Phy. Rev. B* **38**, 12184 (1988).

[16] E. T. Watts, J. Krim and A. Wisdom, *Phy. Rev. B* **41**, 3466 (1990).

[17] J. Krim, D. H. Solina, and R. Chiarello, *Phy. Rev. Lett.* **66**, 181 (1991).

[18] I. K. Snook and W. van Megan, *J. Chem. Phys.* **72**, 2907 (1980).

[19] C. L. Rhykerd Jr., M. Schoen, and D. J. Diestler, *Nature* **330**, 461 (1987).

[20] M. Schoen, D. J. Diestler, and J. H. Cushman, *J. Chem. Phys.* **87**, 5464 (1987).

[21] S. A. Somers and H. T. Davis, *J. Chem. Phys.* **96**, 5389 (1992).

[22] M. Miyahara and K. E. Gubbins, *J. Chem. Phys.* **106**, 2865 (1997).

[23] M. Dijkstra, *J. Chem Phys*. **107**, 3277 (1997).

[24] S. Toxvaerd, *J. Chem. Phys.* **74**, 1998 (1981).

[25] J. J. Magda, M. Tirrell, and H. T. Davis, *J. Chem. Phys.* **83**, 1888 (1985).

[26] Y. Wang, K. Hill, and J. G. Harris, *J. Chem. Phys.* **100**, 3276 (1994).

[27] J. Gao, W. D. Luedtke, and U. Landman, *Phys. Rev. Lett.* **79**, 705 (1997).

[28] J. Gao, W. D. Luedtke, and U. Landman, *J. Phys. Chem. B* **101**, 4013 (1997).

[29] J.-C. Wang and K. A. Fichthorn, *J. Chem. Phys.* **112**, 8252 (2000).

[30] S. Saroja and J.-C. Wang, *Mol. Simul.* **29**, 495 (1993).

[31] T. K. Vanderlick, L. E. Scriven, and H. T. Davis, *J. Chem. Phys.* **90**, 2422 (1989).

[32] C. Lastoskie, K. E. Gubbins, and N. Quirke, *Langmuir* **9**, 2693 (1993).

[33] S. Sarman, *J. Chem. Phys.* **92**, 4447 (1990).

[34] Y. Duda, D. Henderson, A. Trokhymchuk, and D. Wasan, *J. Phys. Chem. B* **103**, 7495 (1999).

[35] O. Reynolds, *Phil. Trans. Roy. Soc.* (Lond.) **177**, 157 (1886).

[36] A. V. Nguyen, *J. Colloid Interface Sci.* **231**, 195 (2000).

[37] H. K. Christenson, *J. Chem. Soc., Faraday Trans. 1* **80**, 1933 (1984).

[38] J. N. Israelachvili, *Intermolecular and Surface Forces*, 2nd ed. Academic Press, San Diego (1992).

[39] H. Yoshizawa and J. N. Israelachvili, *J. Phys. Chem.* **97**, 11300 (1993).

[40] H.-W. Hu, G. A. Carson, and S. Granick, *Phys. Rev. Lett.* **66**, 2758 (1991).

[41] M. Brust, M. Walker, D. Bethell, D. J. Schiffrin, and R. Whyman, J. *Chem. Soc., Chem. Commun.* 801 (1994).

[42] C. P. Collier, T. Vossmeyer, and J. R. Heath, *Annu. Rev. Phys. Chem.* **49,** 371 (1998).

[43] H. S. Nalwa, *Handbook of Nanostructured Materials and Nanotechnology*, Academic Press, San Diego, CA, (2000).

[44] I. Bitsanis, T. K. Vanderlick, M. Tirrel, and H. T. Davis, *J. Chem. Phys.* **89**, 3152 (1988).

[45] L. A. Pozhar and K. E. Gubbins, *J. Chem. Phys.* **99**, 8970 (1993).

[46] M. P. Allen and D. J. Tildesley, *Computer Simulation of Liquids*, Clarendon Press, Oxford (1989).

[47] D. Frenkel and B. Smit, *Understanding Molecular Simulation: From Algorithms to Applications*, Academic Press, San Diego (1996).

[48] X. Bi, *Molecular Dynamics Study of Nanoparticle Self-Assembly*, PhD dissertation, University of Missouri-Rolla (2006).

[49] I. I. Adamenko, A. N. Grigoriev, and Yu. I. Kuzovkov, *J. Mol. Liq.* **120**, 63 (2005).

[50] M. Schoen, D. J. Diestler, and J. H. Cushman, *J. Chem. Phys.* **87**, 5464 (1987).

[51] M. Schoen, J. H. Cushman, D. J. Diestler, and C. L. Rhykerd, Jr., *J. Chem. Phys.* **88**, 1394 (1988).

[52] T. Matsuda, G. D. Smith, R. G. Winkler, and D. Y. Yoon, *Macromolecules* **28**, 165 (1995).

[53] S. K. Kumar, M. Vacatello, and D. Y. Yoon, *J. Chem. Phys.* **89**, 5206 (1988).

[54] I. A. Bitsanis and G. Hadziioannou, *J. Chem. Phys.* **92**, 3827 (1990).

[55] I. A. Bitsanis and C. Pan, *J. Chem. Phys.* **99**, 5520 (1993).

[56] J.-C. Wang and K. A. Fichthorn, , *J. Chem. Phys.* **112**, 8252 (2000).

[57] J.-C. Wang and K. A. Fichthorn, *J. Chem. Phys.* **116**, 410 (2002).

[58] W. G. Hoover, *Phys. Rev. A* **31**, 1695 (1985).

[59] H. J. C. Berendsen, J. P. M. Postma, W. F. van Gunsteren, A. DiNola, and J. R. Haak, *J. Chem. Phys.* **81**, 3684 (1984).

[60] H.-W. Hu, G. A. Carson, and S. Granick, *Phys. Rev. Lett.* **66**, 2758 (1991).

VITA

Ramesh Chembeti was born on March 16, 1983 in Pallamala, Chittoor (District), Andhra Pradesh, India. He received his primary, secondary and intermediate education in Tirupathi, India. He earned a Bachelor of Technology degree in Chemical Engineering from Jawaharlal Nehru Technological University, Hyderabad, India in April 2004.

He enrolled into Master's program at the University of Missouri-Rolla (currently Missouri University of Science and Technology), Rolla in the Spring of 2006 in the department of Chemical Engineering and held a Graduate Research Assistantship throughout the Master's program. He obtained his Master's degree in Chemical Engineering in August 2008.