

PHASE CHANGE STUDIES WITH MOLECULAR DYNAMICS: A COMPUTER SIMULATION

Y. Bayazitoglu*, S. Maruyama**, P. Hos***

* *Professor of Mechanical Engineering, Department of Mechanical Engineering and Materials Science
Rice University, 6100 South Main Street, Houston, TX 77005-1892*

** *Professor of Mechanical Engineering, University of Tokio, 7-3-1 Hongo, Bunkyo-ku, Tokyo 113, Japan*

*** *Graduate Student, Department of Mechanical Engineering and Materials Science
Rice University, 6100 South Main Street, Houston, TX 77005-1892*

ABSTRACT

In this paper we include and explain in detail a computer code used to solve the molecular dynamics simulation of an evaporating droplet. In this code we use the Gear Predictor-Corrector numerical integrator and a truncated Lennard-Jones potential. However, the code can be used for different applications given some minor modifications. The scope of this paper is to give researchers a working code to study molecular dynamics and encourage other researchers to do the same

1 INTRODUCTION

Molecular dynamics simulations could become viable as a tool for analyzing systems on a nanoscale level. This can primarily be attributed to the advanced design of high speed, large memory and/or parallel computers. Molecular dynamics formulation is deterministic and consist of simultaneously solving Newton's equations of motion for each atom, molecule or system of molecules to determine properties of the materials. Molecular dynamic studies are limited to few topics with some success. Among these the most frequent application is to simulate the nanoscale phase change phenomena through an atomic or molecular level analyses. One such application is within the metallurgical industry, an emphasis has been placed on the nanoscale phase change behavior of metals because of their relevance in many diverse nanoscale processes, such as metal powder production via nucleation from a supersaturated vapor, chemical vapor deposition processes, and the crystallization of various polymers. Pound (1952) observed the nucleation of various types of vapors and approximated that the critical nuclei consisted of only between 80 and 100 atoms. Attention has also been focused on the need for understanding the kinetics of nucleating vapor (Reiss, 1952 and Michaels, 1969). Statistical theory predicts a deviation in the behavior of droplets this small from macroscopic or continuum analyses. Hill *et al* (1963) confirmed this notion in his study of nucleating metal vapors by noting that the surface tension could be in substantial error since drops of critical size have such small radii of curvature. Nucleation phenomena are small in spatial extent in that the critical radii of the nuclei have been found to be of the order 10^{-9} m (Lothe and Pound, 1969). Furthermore, the time required for formation of critical nuclei has been documented to be of the order 10^{-10} s (Mandell *et al*,

1976). Molecular Dynamics studies of nucleation are extended to molecular clusters undergoing phase changes with a new interaction potential by Santikara and Bartell (1997), and the crystallization of metastable fluids by Pickering and Snook (1997). Recently, molecular dynamics simulations have been used to investigate the molecular mechanism that governs many heat transfer processes such as the evaporation, condensation and melting at liquid surfaces (Chokappa and Clancy, 1988, Rey *et al* 1992, Yasuoka *et al*, 1994, Matsumoto *et al*, 1994, 1996, Kotake and Aoki, 1996, Tsuruta, Takana, Tamashima and Masuoka, 1996.) The evaporation of droplets studies are needed at supercritical conditions as it relates to combustion in cryogenic rocket motors on a microscale basis as described by Kaltz *et al* (1994). The subcritical evaporation of droplet is modeled by Long *et al* (1997), and the computed evaporation rate compared with the Knudsen aerosol theory. A molecular dynamics simulation is employed by Bhansali, Bayazitoglu and Maruyama (1998) to investigate the interfacial phenomena and to determine properties of an evaporating sodium droplet. The Lennard-Jones potential (Bhansali *et al*, 1996) and oscillatory pair potential based on Levesque *et al* (1985)'s data is studied (Bhansali and Bayazitoglu, 1996). To simulate such a problem the computational code will be presented in the following section.

Molecular dynamics simulations consist of four major phases: 1) the construction of an adequate potential that governs the intermolecular forces acting between the individual particles, 2) the initialization of the simulation and run parameters, 3) the calculation of the molecular trajectories and velocities of each particle during simulation, and 4) the analysis of the trajectories and velocities to determine the physical properties of the system.

On a different note, molecular dynamics studies have not exploded as fast as one would have originally expected. Most of the publications related to our current phase change problem have started to resemble each other very closely. It seems that

this is due to the fact that writing the computer code for solving the problem is such a time consuming and daunting task that most people consider it a milestone alone when getting their code to run.

The computer codes, together with the capabilities of current and future computer technology, have the potential of solving new problems and quantifying the experiments of nano and micro-scale size. We would like to see some more collaboration in the form of making previously produced and tested computer codes available to other researchers such that they can eliminate the pain and agony of trying to write their own code from scratch. This will give people a jump-start, so that they can spend their time expanding the current research or even uncover complete new areas of application. One good example is the book "Molecular Dynamics Simulation" by Haile (1992). Surely many people have put his detailed explanations of the intricacies of the problems involved in writing a molecular dynamics code to good use.

In this paper we will present our analysis of an evaporating droplet along with the code written to solve the problem. We hope other researchers will take this code and use it to their advantage. With some alterations this code can be used to simulate processes such as conduction in thin films or fluid flow in nano channels.

2 MOLECULAR DYNAMICS SIMULATION CODE

In simulation for the intermolecular forces the well-known Lennard-Jones 12-6 potential, ϕ is used,

$$\phi(r_{ij}) = 4\epsilon \left[\left(\frac{\sigma}{r_{ij}} \right)^{12} - \left(\frac{\sigma}{r_{ij}} \right)^6 \right] \quad (1)$$

where r_{ij} is the separation distance between atoms i and j , σ the first zero of the potential is the equilibrium separation parameter (\AA), and ϵ is the potential well depth (J). This potential was selected because of its simplicity, but it could easily be replaced with another potential in the code as long as it represents the physical behavior of the medium (e.g. Bhansali and Bayazitoglu, 1996).

Although the code is initially written to investigate thermal characteristics of an isolated evaporating liquid droplet at various temperatures, it can also be used for other nano-scale problems. A microcanonical ensemble was employed in which the number of particles, the volume, and the total energy of the system were the constrained constants (Rowley, 1994).

The fcc structure is used for simulations in this code. Each atom was arbitrarily assigned a velocity in each coordinate direction (v_{ik}^*) via a random number generator and then scaled to the set point temperature, T_p^* ,

$$v_{ik}^{*new} = v_{ik}^* \sqrt{\frac{T_p^*}{T_{act}^*}} \quad (2)$$

where subscript i corresponds to a particular atom, and subscript k represents the coordinate direction. The instantaneous temperatures are time averaged during the simulation until the steady state is reached. The instantaneous temperature of the system T_{act}^* determined from the equipartition principle.

$$T_{act}^* = \frac{1}{3N} \sum_i \sum_k v_{ik}^* \cdot v_{ik}^* \quad (3)$$

where N is the number of atoms.

The velocities and the temperatures in the simulation were normalized with respect to the standard parameters used in typical soft-sphere models presented in Table 1 and are denoted as such with the * superscript (Thompson *et al*, 1984, Allen and Tildesley, 1987).

Table 1. Reduced Parameters

Quantity	Parameter
Length	$x^* = x/\sigma$ $r^* = r/\sigma$
Pressure	$P^* = (P\sigma^3/\epsilon)$
Temperature	$T^* = (k_B T/\epsilon)$ $k_B = 1.381 \times 10^{-23} \text{ J/K}$
Velocity	$V^* = (v/\sqrt{\epsilon/wm})$
Timestep	$\Delta t^* = (\Delta t/\sigma\sqrt{wm/\epsilon})$
Density	$\rho^* = N\sigma^3/V$
Potential	$\phi^* = \frac{\phi}{N\epsilon}$

Since no dissipative external forces exist and since the potential was assumed pairwise additive, the force F_i (N) on each particle i could then be related to the potential in the following manner

$$\vec{F}_i = -\sum_j \nabla \phi(r_{ij}) = m \frac{\partial^2 \vec{r}_i}{\partial t^2} \quad (4)$$

where ∇ denotes the gradient and wm , (kg) is the mass of the atom, molecule or cluster. F_i is the position vector (Armstrong), and t is the time, (sec). A 5th order Gear predictor-corrector numerical integrator algorithm was used to solve the Newton's equations of motion for each atom. This algorithm was found to have better energy conservation characteristics to the other algorithms considered previously such as the Verlet or Beeman algorithms (Swope *et al*, 1982, Haile, 1994, Amini and Fincham, 1990).

The simulation was governed by the temperature control scheme employed in the simulation. An equilibration period was specified at each set point temperature (T_p^*) during which the velocity was scaled according to equation (2).

Thermal equilibration was monitored by using a nearest neighbor routine that tracked the number of vapor atoms with respect to time. Each atom was classified as vapor, liquid, or surface depending on the number of atoms that are within a sphere of radius 1.5σ centered at that particular atom. An atom was considered vapor if it had 1-2 neighbors, interfacial if it had 3-7 neighbors, and liquid if it had 8 or more neighbor atoms (Maruyama *et al*, 1994). Thermal equilibrium had been reached when the number of vapor atoms had become relatively constant.

The velocity scaling was then terminated and the system was allowed to proceed in a state of constant total energy during which the trajectories and velocities were accumulated for the subsequent determination of the physical properties. The set point temperature was then reset and thermal equilibrium was achieved at the new desired state.

For a system of N atoms, there are $N(N-1)/2$ possible force interactions and the calculation of these interactions is extremely time-consuming. To calculate the force on each atom, for systems in which atoms are subjected to short range forces, a Verlet neighbor list routine was implemented to reduce the computation time (Verlet, 1967, Arnold and Mauser, 1990, Chialvo and Debenedetti, 1991).

Hence, a truncated potential was used such that the force for separation distances greater than a critical cutoff distance, r_c , (A), equals zero, indicating that outside the critical radius atoms have a negligible contribution to the total force on a given atom..

Then for each atom, the routine maintains a list of neighboring atoms that lie within a distance r_l , (A) of that particular atom where typically $r_l = r_c + 0.3\sigma$. The neighbor list for each particle was automatically updated based on a maximum particle displacement criterion as reported by Verlet.

In Fig 1 we present the flow chart of the code. The code is written in FORTRAN77 and the main and the subroutines are individually explained.

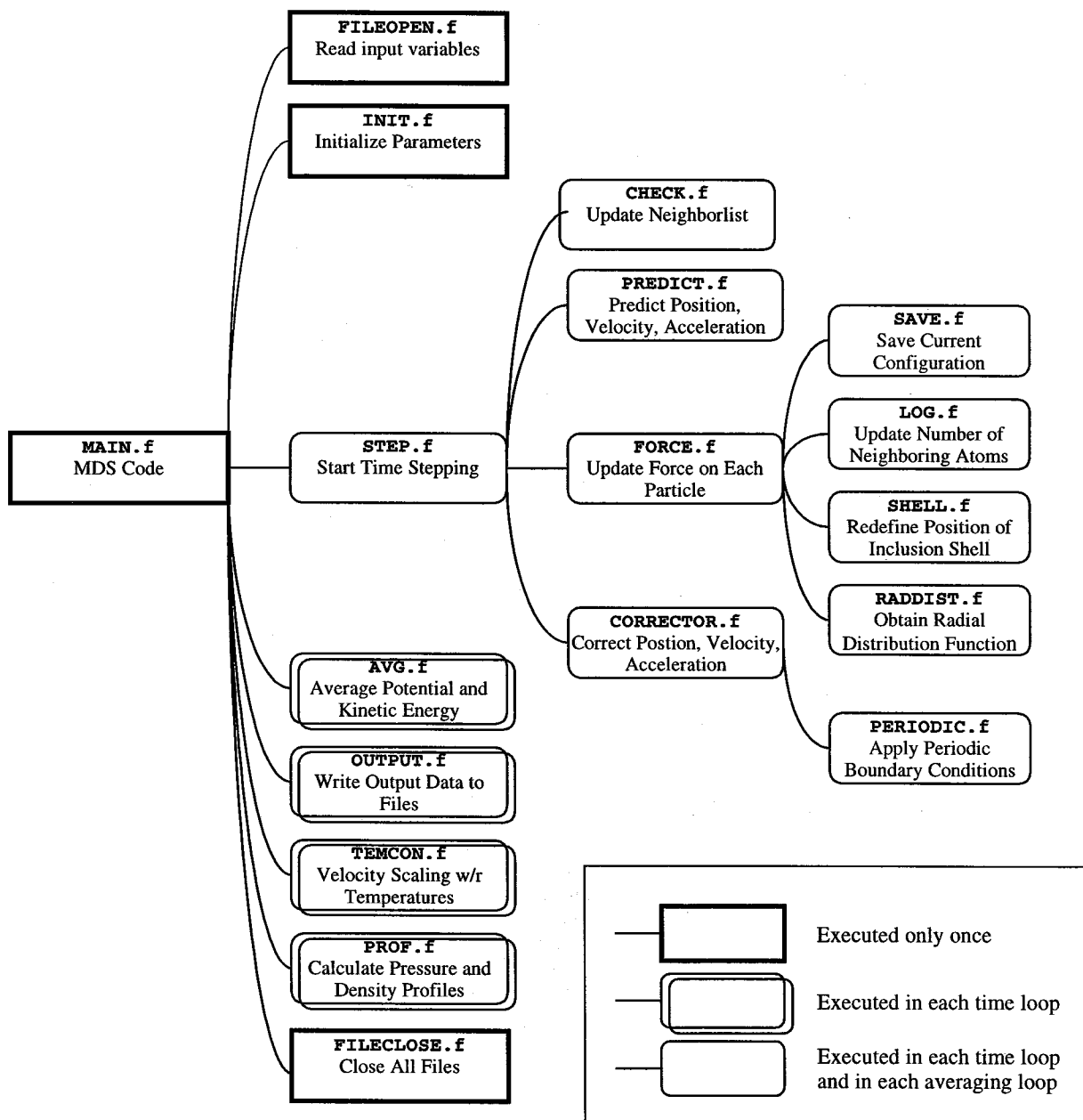


Figure 1. Flowchart

2.1 Main Routine

```

c*****
c
c
c          MAIN BODY
c
c*****
integer nmax,maxnab
parameter (nmax=865, maxnab=nmax*500)
integer i,iloc,inte,intlg,intp,inttc,intv,irep,j,list,
+ lnay,nloc,nmol,nrep,ntc,point,upcnt,rknt
integer lsti (nmax*(nmax-1)/2),lstj (nmax*(nmax-1)/2)
double precision enesum,plr,ulr,ve,velsq,vsqsum,wu
integer kavg,kprnt

c RDF = radial distance function
integer ngofr,rdels
double precision gr,rdel
double precision bk,dt,dts,eps,ffc,fnb,pi,rcut,rlist,
+ rxo,ryo,rzo,sig,time,tini,tins,uls,
+ virsum,vl,vls,wm
double precision x(nmax,3),v(nmax,3),xi(nmax,3)
double precision accx,accy,accz,alp0,alp1,alp2,alp3,alp4,
+ bx,by,bz,cx,cy,cz
logical lpo,lvo,update

common /one/ lsti,lstj,v,x
common /block1/ rxo(nmax),ryo(nmax),rzo(nmax)
common /block2/ list(maxnab),point(nmax)
common /gear/ accx(nmax),accy(nmax),accz(nmax),
+ bx(nmax),by(nmax),bz(nmax),
+ cx(nmax),cy(nmax),cz(nmax)
common /correct/ alp0,alp1,alp2,alp3,alp4

c RDF
common /rdf/ gr(500),rdel
common /rdf2/ rdels,ngofr(500)

c
c Open data files for formatting
c
call fileopen
rknt = 0

c
c Calls for the initial condition subroutine
c
call init(bk,dt,dts,enesum,eps,ffc,fnb,inte,intlg,intp,
+ inttc,intv,kavg,kprnt,lnay,nloc,nmol,nrep,ntc,
+ pi,plr,rcut,rlist,sig,time,tini,tins,ulr,uls,
+ upcnt,update,virsum,vl,vls,vsqsum,wm,xi)

c
c Start stepping in time
c Print initial conditions
c
write(13,'(3I7)') 1,nmol,nrep/intp+1
write(13,'(3F7.2)') vl*1.0D10,time*1.0D12,dt*nloc*intp*1.0D12
do 10 i = 1,nmol
write(13,'(3F7.2)') (x(i,j)*1.0D10*sig,j=1,3)
10 continue

c
c TOP OF THE LOOP
c
c Update time of the simulation
c
nrep = number of time steps general loop is executed = 2000
do 100 irep = 1,nrep
open(9,file='times.dat')
write(9,*) irep
close(9)
time = time+dts*nloc

c
c MOVE MOLECULES
c
c Call subroutine "step". Atoms are moved for nloc times from
c their current configurations. This is called the local loop
c nloc = 10
c
do 110 iloc = 1,nloc
call step(bk,dts,eps,ffc,fnb,iloc,irep,lnay,nloc,
+ nmol,rcut,rlist,sig,update,upcnt,ve,
+ virsum,vl,vls,wm,wu,uls,pi,xi,rknt)
110 continue

c
c Check whether logicals are true or false
c lpo = Controls position vector output
c lvo = Controls velocity vector output
c
lpo = .false.
if(mod(irep,intp).eq.0) lpo = .true.
lvo = .false.
if(mod(irep,intv).eq.0) lvo = .true.
if(mod(irep,kavg).eq.0) then
call avg(enesum,nmol,velsq,vsqsum,v,ve,virsum,wu)
end if
if(mod(irep,kprnt).eq.0) then
call output(bk,enesum,eps,irep,kavg,lnay,lpo,lvo,nloc,
+ nmol,plr,sig,time,ulr,velsq,vsqsum,
+ v,ve,virsum,vls,wm,wu,x)
end if
if((mod(irep,inttc).eq.0).and.(irep.le.ntc)) then
c Subroutine TEMCON corrects the velocities with temperature.

```

```

c This allows the system to equilibrate at a certain
c temperature.
call temcon(irep,nmol,tins,v)
end if
if(mod(irep,inte).eq.0)then
c PROF is called once at he end of the code. ( inte=2000)
call prof(bk,eps,fnb,irep,lpo,lvo,nmol,pi,rcut,
+ sig,time,tini,tins,virsum,vl,vls,wm)
endif

c-----
c BOTTOM OF THE LOOP
c-----
100 continue

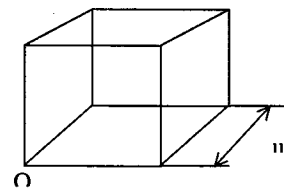
c-----
c OUTPUT FINAL CONDITION (12)
c-----
c Writes the final position vectors to a file
c-----
write(12,*) nmol,vls,uls,time
do 777 i = 1,nmol
write(12,*) x(i,1),x(i,2),x(i,3)
777 continue
do 666 i = 1,nmol
write(12,*) v(i,1),v(i,2),v(i,3)
666 continue

c-----
c CLOSE FILES
c-----
c Call to subroutine "fileclose" and verify that the program
c terminated normally
c-----
call fileclose
write(*,*) '-- program finished normally --'
stop
end

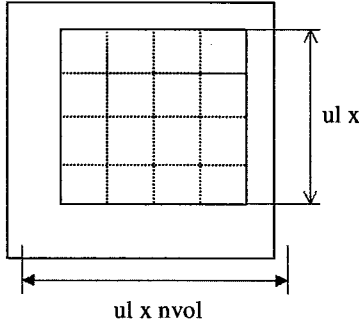
```

2.2 Subroutine INIT

This subroutine is called by the main program once at the beginning of the code. It initializes the variables. Initial positions of atoms are arranged based on Face Centered Cubic structure. 8 Atoms on the corners are common to 8 cubes and 6 atoms at the center of the faces are common to 2 cubes. So, the number of atoms for a unit cell can be calculated as follows:
 (Number of atoms on the corners) x (0.125) + (number of atoms at the faces) x (0.5) = 8 x (0.125) + 6 x (0.5) = 4



if $O(0,0,0) \Rightarrow$ The initial coordinates of 4 atoms in a cell will be: $(0, 0, 0)$, $(ul/2, ul/2, 0)$, $(0, ul/2, ul/2)$, $(ul/2, 0, ul/2)$
 For this analysis the number of unit cells in a crystal is equal to $(nuni)^3$. So the number of atoms in a unit crystal is $4 \cdot (nuni)^3$, where $nuni = 4$
 So the length of a crystal = $(ul) \times (nuni)$
 then we define control volume as a cube having a length of vl where $vl = (ul) \times (nvol)$, where $nvol = 6$



To initialize atoms, initial distance from an atom on the outside boundary to the periodic boundary is defined as:

$$\text{base} = \frac{1}{2}(\text{vl} - (\text{nuni}) \times (\text{ul})) \quad (5)$$

After the initial positions are set, they are nondimensionalized with respect to σ .

Initial velocities

Each atom is assigned to an initial velocity having a magnitude of 1.0

$$\text{bc} = \text{random magnitude} = -1.0 \leq \text{bc} \leq 1.0$$

$$\text{a} = \text{Random angle}$$

$$\text{bs} = \sqrt{1 - \text{bc}^2}$$

The magnitude of velocity,

$$|V_i| = \sqrt{(\text{bc} \cdot \sin a)^2 + (\text{bc} \cdot \cos a)^2 + \text{bs}^2} = 1.0 \quad (6)$$

The components of velocity are going to be expressed as follows in the code

$$\vec{V}_n = V(n,1) \vec{i} + V(n,2) \vec{j} + V(n,3) \vec{k} \quad (7)$$

$$V(n,1) = \text{bc} \sin(a)$$

$$V(n,2) = \text{bc} \cos(a)$$

$$v(n,3) = \sqrt{1 - \text{bc}^2}$$

Cancellation of the total translation

To obtain zero total linear momentum the velocities are scaled by removing translation from the velocities.

Total translation in x, y, z -directions are $V1 = \sum_{i=1}^{\text{nmol}} V(i,1)$,

$V2 = \sum_{i=1}^{\text{nmol}} V(i,2)$ and $V3 = \sum_{i=1}^{\text{nmol}} V(i,3)$. So, Average x, y, z -

translations / atom are $\frac{V1}{\text{nmol}}$, $\frac{V2}{\text{nmol}}$, $\frac{V3}{\text{nmol}}$ respectively.

Velocities after translation, can then be written as

$$V(i,1) = V(i,1) - \frac{V1}{\text{nmol}}, \quad V(i,2) = V(i,2) - \frac{V2}{\text{nmol}},$$

$$V(i,3) = V(i,3) - \frac{V3}{\text{nmol}}$$

Cancellation of the total rotation

Position of an atom :

$$x(i,1)\vec{i} + x(i,2)\vec{j} + x(i,3)\vec{k} \quad (8)$$

Velocity of an atom :

$$\vec{V}_i = V(i,1) \vec{i} + V(i,2) \vec{j} + V(i,3) \vec{k} \quad (9)$$

The components of the distance of an atom from the center of the control volume:

$$r_1 = x(i,1) - \frac{\text{vl}}{2}, \quad r_2 = x(i,2) - \frac{\text{vl}}{2}, \quad r_3 = x(i,3) - \frac{\text{vl}}{2} \quad (10)$$

then we can write angular rotation about x, y, z axes in the following form

$$O1 = \sum_{i=1}^{\text{nmol}} [r_2 \times V(i,3) - r_3 \times V(i,2)]$$

$$O2 = \sum_{i=1}^{\text{nmol}} [r_3 \times V(i,1) - r_1 \times V(i,3)] \quad (11)$$

$$O3 = \sum_{i=1}^{\text{nmol}} [r_1 \times V(i,2) - r_2 \times V(i,1)]$$

Average angular velocities are calculated as:

$$O1 = \frac{O1}{\sum_{i=1}^{\text{nmol}} |\vec{r}_i|^2}, \quad O2 = \frac{O2}{\sum_{i=1}^{\text{nmol}} |\vec{r}_i|^2}, \quad O3 = \frac{O3}{\sum_{i=1}^{\text{nmol}} |\vec{r}_i|^2} \quad (12)$$

Velocities after removal of rotation are

$$V(i,1) = V(i,1) - (O2 \times r_3 - O3 \times r_2)$$

$$V(i,2) = V(i,2) - (O3 \times r_1 - O1 \times r_3) \quad (13)$$

$$V(i,3) = V(i,3) - (O1 \times r_2 - O2 \times r_1)$$

Now, the velocities should be adjusted to match the temperature of the system.

The total kinetic energy per unit mass is given as

$$\text{KEG} = \sum_{i=1}^{\text{nmol}} V(i,1)^2 + V(i,2)^2 + V(i,3)^2 \quad (14)$$

The kinetic energy per atom ($\text{ek} = \text{KE} / \text{nmol}$) can also be written in terms of temperature:

$$\text{ek} = \frac{3k_b T}{\text{wm}} \quad (15)$$

so, we can write the scaling factor as,

$$\alpha = \sqrt{\frac{3k_b T}{\text{wm} \cdot \text{ek}}} \quad (16)$$

And then velocities are nondimensionalized with $\sqrt{\frac{\text{wm}}{\epsilon}}$

```

c*****
c
  subroutine init(bk,dt,dts,enesum,eps,ffc,fnb,inte,intlg,
+ intp,inttc,intv,kavg,kprnt,lnay,nloc,nmol,nrep,ntc,
+ pi,plr,rcut,rlist,sig,time,tini,tins,ulr,uls,upcnt,
+ update,virsum,v1,vls,vsqsum,wm,xi)
c*****
c
  implicit none
  integer nmax
  parameter (nmax=865)
  integer i,im,inte,intlg,intp,inttc,intv,j,k,kavg,kprnt,
+ lnay,nloc,nmol,nrdm,nrep,ntc,nuni,nvol,upcnt
  integer lsti(nmax*(nmax-1)/2),lstj(nmax*(nmax-1)/2)
  double precision x(nmax,3),v(nmax,3),xi(nmax,3)
  double precision ai0,an,ang,base,bc,bk,bs,coef,dt,dts,ek,
+ enesum,eps,ffc,fnb,o1,o2,o3,pi,plr,r1,r2,
+ r3,ra2,randf,rcut,rlist,sig,time,tini,tins,ul,
+ ulr,uls,v1,v2,v3,virsum,v1,vls,vr1,vr2,vr3,
+ vsqsum,wm
  double precision accx,accy,accz,alp0,alp1,alp2,alp3,alp4,
+ bx,by,bz,cx,cy,cz
c RDF

```

```

integer ngofr, nrdels
double precision gr, rdcl
logical update
common /one/ lsti, lstj, v, x
common /gear/ accx(nmax), accy(nmax), accz(nmax),
+          bx(nmax), by(nmax), bz(nmax),
+          cx(nmax), cy(nmax), cz(nmax)
common /correct/ alp0, alp1, alp2, alp3, alp4
c RDF
common /rdf/ gr(500), rdcl
common /rdE2/ nrdels, ngofr(500)
c -----
c Specify the constants
c -----
pi = acos(-1.0)
bk = 1.381D-23
an = 6.022D23
c -----
c Specify the parameters of Sodium
c -----
wm = 22.99D-3/an
sig = 3.240D-10
eps = 8.27D-21
c -----
c read CONDITIONS (10)
c -----
read(10,*) nuni
read(10,*) nvol
read(10,*) tini
read(10,*) vl
read(10,*) dt
read(10,*) nrep
read(10,*) nloc
read(10,*) intp
read(10,*) intv
read(10,*) inte
read(10,*) inttc
read(10,*) intlq
read(10,*) ntc
read(10,*) nrdm
read(10,*) fnb
read(10,*) ffc
read(10,*) kavg
read(10,*) kprnt
read(10,*) lnay
c -----
c INITIALIZE POSITION AND VELOCITY
c -----
c READ PREVIOUS RESULT (11)
c -----
read in data from file "pre.dat" to continue previous
simulation if it exists
read(11,*,end=100) nmol,vls,uls,time
do 900 i = 1,nmol
  read(11,*,end=100) x(i,1),x(i,2),x(i,3)
900  continue
do 901 i = 1,nmol
  read(11,*,end=100) v(i,1),v(i,2),v(i,3)
901  continue
write(*,*) 'Continuing previous calculation.'
goto 192
c -----
c START NEW CALCULATION
c -----
100 continue
nmol = 4*nuni**3
ul = vl/nvol
uls = ul/sig
vls = vl/sig
c -----
c POSITION (x)
c -----
base = 0.5*(vl-nuni*ul)
do 110 i = 0, nuni-1
  do 110 j = 0, nuni-1
    do 110 k = 0, nuni-1
      im = (k+j*nuni+i*nuni**2)*4
      x(im+1,1) = ul*(i+1.0/4)+base
      x(im+1,2) = ul*(j+1.0/4)+base
      x(im+1,3) = ul*(k+1.0/4)+base
      x(im+2,1) = ul*(i+3.0/4)+base
      x(im+2,2) = ul*(j+1.0/4)+base
      x(im+2,3) = ul*(k+3.0/4)+base
      x(im+3,1) = ul*(i+1.0/4)+base
      x(im+3,2) = ul*(j+3.0/4)+base
      x(im+3,3) = ul*(k+3.0/4)+base
      x(im+4,1) = ul*(i+3.0/4)+base
      x(im+4,2) = ul*(j+3.0/4)+base
      x(im+4,3) = ul*(k+1.0/4)+base
    110 continue
  110 continue
110 continue
c -----
c Nondimensionalize positions now
c -----
do 115 i = 1, nmol
  x(i,1) = x(i,1)/sig
  x(i,2) = x(i,2)/sig
  x(i,3) = x(i,3)/sig
  xi(i,1) = x(i,1)
  xi(i,2) = x(i,2)
  xi(i,3) = x(i,3)
115 continue
do 116 i = 1, nmol
  write(34,134) xi(i,1),xi(i,2),xi(i,3)
116 continue
134 format(f10.4,2x,f10.4,2x,f10.4)

```

```

c -----
c VELOCITY (v)
c -----
do 120 i = 1, nmol
  ang = randf(nrdm)*pi*2.0
  bc = 1.0-2*randf(nrdm)
  bs = dsqrt(1.0-bc*bc)
  v(i,1) = bc*sin(ang)
  v(i,2) = bc*cos(ang)
  v(i,3) = bs
120 continue
c -----
c CANCEL THE TOTAL TRANSLATION
c -----
v1 = 0.0
v2 = 0.0
v3 = 0.0
do 130 i = 1, nmol
  v1 = v1+v(i,1)
  v2 = v2+v(i,2)
  v3 = v3+v(i,3)
130 continue
v1 = v1/nmol
v2 = v2/nmol
v3 = v3/nmol
do 140 i = 1, nmol
  v(i,1) = v(i,1)-v1
  v(i,2) = v(i,2)-v2
  v(i,3) = v(i,3)-v3
140 continue
c -----
c CANCEL THE TOTAL ROTATION
c -----
ai0 = 0.0
o1 = 0.0
o2 = 0.0
o3 = 0.0
do 150 i = 1, nmol
  r1 = x(i,1)-0.5*v1
  r2 = x(i,2)-0.5*v1
  r3 = x(i,3)-0.5*v1
  ra2 = r1*r1 + r2*r2 + r3*r3
  ai0 = ai0+ra2
  o1 = o1+(r2*v(i,3)-r3*v(i,2))
  o2 = o2+(r3*v(i,1)-r1*v(i,3))
  o3 = o3+(r1*v(i,2)-r2*v(i,1))
150 continue
o1 = o1/ai0
o2 = o2/ai0
o3 = o3/ai0
do 160 i = 1, nmol
  r1 = x(i,1)-0.5*v1
  r2 = x(i,2)-0.5*v1
  r3 = x(i,3)-0.5*v1
  vr1 = o2*r3-o3*r2
  vr2 = o3*r1-o1*r3
  vr3 = o1*r2-o2*r1
  v(i,1) = v(i,1)-vr1
  v(i,2) = v(i,2)-vr2
  v(i,3) = v(i,3)-vr3
160 continue
c -----
c ADJUST THE TEMPERATURE
c -----
Velocities will now be nondimensional
c -----
ek = 0.0
do 170 i = 1, nmol
  ek = ek + v(i,1)**2 + v(i,2)**2 + v(i,3)**2
170 continue
ek = ek/nmol
coef = dsqrt(3.0*bk*tini/(wm*ek))
do 180 i = 1, nmol
  v(i,1) = v(i,1)*coef*(wm/eps)**0.5
  v(i,2) = v(i,2)*coef*(wm/eps)**0.5
  v(i,3) = v(i,3)*coef*(wm/eps)**0.5
180 continue
write(*,*) 'Starting new calculation.'
time = 0.0
c -----
c Initialize parameters for Gear algorithm
c -----
192 do 195 i = 1, nmol
  accx(i) = 0.
  accy(i) = 0.
  accz(i) = 0.
  bx(i) = 0.
  by(i) = 0.
  bz(i) = 0.
  cx(i) = 0.
  cy(i) = 0.
  cz(i) = 0.
195 continue
alp0 = 19./120.
alp1 = 3./4.
alp2 = 1.
alp3 = 1./2
alp4 = 1./12.
200 continue
c -----
c Initialize parameters for averaging properties
c -----
enesum = 0.
virsum = 0.
vsqsum = 0.

```

```

c-----
c LIST VECTORS (lsti,lstj)
c-----
      k = 0
      do 210 i = 1,nmol-1
        do 210 j = i+1,nmol
          k = k+1
          lsti(k) = j-i
          lstj(k) = j
        210 continue
c-----
c Specify parameters for the
c Verlet Neighbor routine
c-----
      update = .true.
      rcut = 3.5
      rlist = 3.8
c-----
c long-range correction factors
c-----
      plr = 96.*pi*nmol/(3.*vls**3.)
      + * (0.5/(3.*rcut**3.) - 1./(9.*rcut**9.))
      ulr = 8.*pi*(nmol/vls**3.)
      + * (1./(9.*rcut**9.) - 1./(3.*rcut**3.))
c-----
c Nondimensional variables
c-----
      dts = dt/(sig*(wm/eps)**0.5)
      tins = bk*tini/eps
c-----
c Initialize and specify params
c for rad. dist. fun.
c-----
      rdel = 0.025
      nrdels = vls/(2*rdel) - 1
      do 951 i = 1,300
        gr(i) = 0.
        ngofr(i) = 0
      951 continue
      return
      end

```

2.3 Subroutine STEP

This subroutine is used to move molecules. It is called by main program 10 times for each time step.

First of all we check if the neighbor list needs to be updated. This is going to affect the number of atoms interacting with the atom being considered. In other words, the force domain is going to change. To check this we calculate the displacements from the current configuration and then compare those values with 0.3σ (subroutine CHECK).

Then, we start Gear predictor-corrector scheme. First, we call subroutine PREDICT to predict the values of positions, velocities, accelerations, third and fourth derivatives of the positions. The way the method works:

1. Use Taylor expansions to predict the values of desired quantities at the next time step.
2. Calculate the force using predicted positions. Evaluate the acceleration from Newton's second law. Find the difference between this acceleration and acceleration from the predictor step.
3. Use this difference as a correction factor to correct the positions and velocities at the next time step.

```

c*****
c
c      subroutine step(bk,dts,eps,ffc,fnb,iloc,irep,lnay,nloc,
+      nmol,rcut,rlist,sig,update,upcnt,ve,
+      virsum,vl,vls,wm,wu,uis,pi,xi,rknt)
c
c*****
      implicit none
      integer nmax,maxnab
      parameter (nmax=865,maxnab=nmax*500)
      integer i,iloc,irep,ixs,j,list,lnay,nloc,nmol,nxs,
+      point,upcnt,rknt
      integer lsti(nmax*(nmax-1)/2),lstj(nmax*(nmax-1)/2)
      double precision fm(nmax,3),x(nmax,3),v(nmax,3),xi(nmax,3)
      double precision bk,dts,dtsq,dt2,ek,eps,ffc,fnb,rcut,rlist,
+      rxo,ryo,rzo,scl,sig,ve,virsum,vl,vli,vls,

```

```

+      wm,wu,xsl,xs2,xs3,
+      uis,pi
c RDP
      integer ngofr,nrdels
      double precision gr,rdel
      logical update

      common /one/ lsti,lstj,v,x
      common /block1/ rxo(nmax),ryo(nmax),rzo(nmax)
      common /block2/ list(maxnab),point(nmax)
c RDP
      common /rdf/ gr(500),rdel
      common /rdf2/ nrdels,ngofr(500)
      vli = 1.0/vls
      dtsq = dts*dts/2.
      dt2 = dts/2.
c-----
c Make call to check if the Verlet
c neighbor listneeds updating
c-----
      call check(iloc,irep,nmol,rcut,rlist,update,x,upcnt)
c-----
c Predictor values for GPC
c-----
      call predict(dts,nmol,v,x)
c Now, we have the predicted value of the position of each
c atom that will be used in the force calculations.
c-----
c call force to update force on each particle
c-----
      call force(fm,fnb,iloc,irep,lnay,nloc,nmol,rcut,rlist,sig,
+      ve,vl,vls,wu,update,x)
c-----
c Corrector values for GPC
c-----
c After force calculation positions,velocities and higher
c order derivatives of the positions should be corrected.
c CORRECTOR is used.
      call corrector(dts,fm,nmol,v,vls,x,iloc,irep)
c A face centering parameter, ffc, is used that defines a box
c within the control box
c which is used to encompass only the liquid drop. This is done
so that the new
c center of mass is fixed, based solely on the liquid droplet.
      xs1=0.0
      xs2=0.0
      xs3=0.0
      nxs=0
      scl=ffc*vls*0.5
      do 240 i=1,nmol
        ix= idint(dsign(0.5D0,scl-dabs(x(i,1)-0.5*vls))+0.51)
        + *idint(dsign(0.5D0,scl-dabs(x(i,2)-0.5*vls))+0.51)
        + *idint(dsign(0.5D0,scl-dabs(x(i,3)-0.5*vls))+0.51)
c The lines provide a running total of what atoms fall inside
c the ffc box at each time step. This helps to define the
c center of mass of the droplet.
        xs1=xs1+x(i,1)*ix
        xs2=xs2+x(i,2)*ix
        xs3=xs3+x(i,3)*ix
        nxs=nxs+ix
      240 continue
      ek = 0.0
      do 290 i = 1,nmol
        do 290 j = 1,3
          ek = ek+v(i,j)*v(i,j)
        290 continue
c-----
c FIX THE TOTAL CENTER OF MASS (x)
c-----
c Locate the distance in each coordinate, x, y, z that the
c drop is off center with respect to the center of the control
c volume.
      xs1 = xs1/nxs - 0.5*vls
      xs2 = xs2/nxs - 0.5*vls
      xs3 = xs3/nxs - 0.5*vls
c So, the repositioning of each atom is completed based on the
c off center distance.
      do 300 i = 1,nmol
        x(i,1) = x(i,1) - xs1
        x(i,2) = x(i,2) - xs2
        x(i,3) = x(i,3) - xs3
        if ((x(i,1).lt.0).or.(x(i,2).lt.0).or.(x(i,3).lt.0)) then
          call periodic(x(i,1),x(i,2),x(i,3),vls)
        end if
      300 continue
      return
      end

```

2.4 Subroutine CHECK

This subroutine is used to check the updating conditions of the Verlet neighbor list. It is called by the subroutine, STEP.

To run the code more efficiently a truncated potential is used. Because, LJ (12,6) potential gets very close to zero after $r = 2.5\sigma$. In this study cutoff distance was selected as 3.5σ . And for each atom i , the method maintains a list of

neighboring atoms that lie within a distance r_L of i ; so the list identifies those atoms that contribute to the force on atom i . The same neighbor list used over several consecutive time steps, and it is updated every 10 time steps. The list distance is slightly larger than r_c so that j atoms can cross r_c and still be properly considered in evaluating the force on i . For our purposes,

$$r_c = 3.5\sigma \quad \text{and} \quad r_L = r_c + 0.3\sigma = 3.8\sigma \quad (17)$$

```

c*****
c
c      subroutine check(iloc,irep,nmol,rcut,rlist,update,x,upcnt)
c
c*****
c      decides whether the Verlet list needs to be reconstructed.
c      this subroutine is called to set update before every call
c      to force.

      implicit none
      integer nmax
      parameter (nmax=865)
      integer i,iloc,irep,nmol,upcnt
      double precision dispmx,rcut,rlist,rxo,ryo,rzo
      double precision x(nmax,3)
      common /block1/ rxo(nmax),ryo(nmax),rzo(nmax)
      logical update
      dispmx = 0.0

      do 30 i = 1,nmol
         dispmx = dmax1(dabs(x(i,1) - rxo(i)),dispmx)
         dispmx = dmax1(dabs(x(i,2) - ryo(i)),dispmx)
         dispmx = dmax1(dabs(x(i,3) - rzo(i)),dispmx)
      30 continue
      dispmx = 2.0 *dsqrt(3.0*dispmx**2.)
      update = (dispmx.gt.(rlist - rcut))
c
      rlist - rcut = 0.3*
      if (update) then
         upcnt=upcnt+1
         write(24,*) irep,iloc,upcnt
      end if
      return
      end

```

2.5 Subroutine PREDICT

This subroutine is the first step of Gear Algorithm. Positions, velocities, accelerations and higher order derivatives of the positions are predicted for the next step. It is called in the subroutine STEP.

```

c*****
c
c      subroutine predict(dts,nmol,v,x)
c
c*****
c      calculate predicted values prior to force evaluation.
      implicit none
      integer nmax
      parameter (nmax=865)
      integer i,nmol
      double precision dts,v(nmax,3),x(nmax,3)
      double precision accx,accy,accz,bx,by,bz,cx,cy,cz,
      double precision c1,c2,c3,c4
      common /gear/ accx(nmax),accy(nmax),accz(nmax),
      +          bx(nmax),by(nmax),bz(nmax),
      +          cx(nmax),cy(nmax),cz(nmax)
      c1 = dts
      c2 = c1*dts/2.
      c3 = c2*dts/3.
      c4 = c3*dts/4.
c      Predicted values are obtained by expanding Taylor series.
      do 100 i = 1,nmol
         x(i,1)= x(i,1)+c1*v(i,1)+c2*accx(i) + c3*bx(i) + c4*cx(i)
         x(i,2)= x(i,2)+c1*v(i,2)+c2*accy(i) + c3*by(i) + c4*cy(i)
         x(i,3)= x(i,3)+c1*v(i,3)+c2*accz(i) + c3*bz(i) + c4*cz(i)
         v(i,1) = v(i,1) + c1*accx(i) + c2*bx(i) + c3*cx(i)
         v(i,2) = v(i,2) + c1*accy(i) + c2*by(i) + c3*cy(i)
         v(i,3) = v(i,3) + c1*accz(i) + c2*bz(i) + c3*cz(i)
         accx(i) = accx(i) + c1*bx(i) + c2*cx(i)
         accy(i) = accy(i) + c1*by(i) + c2*cy(i)
         accz(i) = accz(i) + c1*bz(i) + c2*cz(i)
         bx(i) = bx(i) + c1*cx(i)
         by(i) = by(i) + c1*cy(i)
         bz(i) = bz(i) + c1*cz(i)
      100 continue
      return
      end

```

2.6 Subroutine FORCE

This is the intermediate step of the Gear Algorithm. FORCE calculates the intermolecular force on each atom. We assume that the interaction energy among N atoms is a sum of isolated two body contributions, which is also called pairwise additivity. FORCE is called in the subroutine STEP. The positions are determined in subroutine PREDICT. To calculate the forces, one should find the distances between two atoms. r_1 , r_2 , r_3 are the x , y , z distances between two atoms respectively.

$$(r_{abs})_{ij} = \sqrt{r_1^2 + r_2^2 + r_3^2} \quad (18)$$

To calculate the intermolecular forces we use Lennard-Jones (12,6) potential, which is given by the expression below.

$$\phi_{ij} = 4\epsilon \left[\left(\frac{\sigma}{r_{abs_{ij}}} \right)^{12} - \left(\frac{\sigma}{r_{abs_{ij}}} \right)^6 \right] \quad (19)$$

then the force that atom i exerts on j is =

$$df_{ij} = -\frac{d\phi_{ij}}{dr} = \frac{24\epsilon}{r_{abs}} \left[2 \left(\frac{\sigma}{r_{abs}} \right)^6 - 1 \right] \left(\frac{\sigma}{r_{abs}} \right)^6 \quad (20)$$

x , y , z components of the force i exerts on j are respectively:

$$\begin{aligned} df_{ij}(i,j,1) &= df_{ij} \frac{r_1}{(r_{abs})_{ij}}, & df_{ij}(i,j,2) &= df_{ij} \frac{r_2}{(r_{abs})_{ij}}, \\ df_{ij}(i,j,3) &= df_{ij} \frac{r_3}{(r_{abs})_{ij}} \end{aligned} \quad (21)$$

Calculate the total force on each particle:

$$\begin{aligned} f_m(j,1) &= \sum_{i=1}^{nmol} df_{ij}(i,j,1), & f_m(j,2) &= \sum_{i=1}^{nmol} df_{ij}(i,j,2), \\ f_m(j,3) &= \sum_{i=1}^{nmol} df_{ij}(i,j,3) \end{aligned} \quad (22)$$

The total force on atom j is then

$$\sqrt{f_m(j,1)^2 + f_m(j,2)^2 + f_m(j,3)^2} \quad (23)$$

```

c*****
c      subroutine force(fm,fnb,iloc,irep,lnay,nloc,nmol,rcut,rlist
c      +
c*****
c      subroutine FORCE is called to calculate the force on each
c      molecule
      implicit none
      integer nmax,maxnab
      parameter (nmax=865,maxnab=nmax*500)
      integer i,iloc,irep,j,jbeg,jend,jnab,list,lnay,nlist,nloc,
      +          nmol,point,nij(nmax*(nmax-1)/2),nmb(nmax)
      double precision fm(nmax,3),x(nmax,3)
      double precision fij,fnb,fxi,fxij,fyi,fyij,fzi,fzij,rcut,
      +          rlist,rx1,rxij,rxo,ry1,ryij,ryo,rzi,rzij,
      +          rzo,sig,sr2,sr6,ve,vij,vl,vli,wij,wu
      double precision vls,rabi,rabs
c      RDP
      integer ngofr,nrdels
      double precision gr,rdel
      common /block1/ rxo(nmax),ryo(nmax),rzo(nmax)
      common /block2/ list(maxnab),point(nmax)
      common /rdf/ gr(500),rdel
      logical update
      vli = 1./vls
c      zero the forces
      do 10 i = 1,nmol
         fm(i,1) = 0.
         fm(i,2) = 0.
         fm(i,3) = 0.
      10 continue
c      Zero the virial force, and neighbors
      do 50 i = 1,nmol
         nmb(i) = 0

```



```

50 continue
   ve = 0.
   wu = 0.
c first of all we look at the result of subroutine CHECK. If the
c neighbor list needs to
c be updated we save the current configuration and reconstruct
c the neighbor list. to save the current configuration,
c subroutine SAVE is called
   if (update) then
     call save(fm,nmol,x)
     nlist = 0
     do 100 i = 1,nmol-1
       point(i) = nlist + 1
       rxi = x(i,1)
       ryi = x(i,2)
       rzi = x(i,3)
       fxi = fm(i,1)
       fyi = fm(i,2)
       fzi = fm(i,3)

       do 99 j = i+1,nmol
         rxij = rxi - x(j,1)
         ryij = ryi - x(j,2)
         rzij = rzi - x(j,3)
         rxij = rxij - dnint(rxij*vli)*vls
         ryij = ryij - dnint(ryij*vli)*vls
         rzij = rzij - dnint(rzij*vli)*vls
         rabs = dsqrt(rxij*rxij + ryij*ryij + rzij*rzij)
         rabi = 1/rabs
         if (rabs.lt.rlist) then
           nlist = nlist + 1
           list(nlist) = j
c then we update neighbor list by calling subroutines LOG, SHELL,
c RADDIST.
         if((mod(irep,lnay).eq.0).and.(mod(iloc,nloc).eq.0))then
           call log(fnb,nij,nlist,rxij,ryij,rzij)
           call shell(rxij,ryij,rzij)
           call raddist(irep,lnay,nmol,sig,vl)
         end if
         if (nlist.eq.maxnab) stop 'list too small'
         if (rabs.lt.rcut) then
           sr2 = rabi**2
           sr6 = sr2*sr2*sr2
c Nondimensional potential
           vij = 4*sr6*(sr6 - 1.)
c Nondimensional force
           wij = 48*sr6*(sr6 - 0.5)
           ve = ve + vij
           wu = wu + wij
           fij = wij*rabi**2
           fxij = rxij*fij
           fyij = ryij*fij
           fzij = rzij*fij
           fxi = fxi + fxij
           fyi = fyi + fyij
           fzi = fzi + fzij
           fm(j,1) = fm(j,1) - fxij
           fm(j,2) = fm(j,2) - fyij
           fm(j,3) = fm(j,3) - fzij
         end if
       end if
     99 continue
       fm(i,1) = fxi
       fm(i,2) = fyi
       fm(i,3) = fzi
     100 continue
       point(nmol) = nlist + 1
     else
c If neighbor list does not need to be updated, we don't
c the current configuration. (But again We use predicted
c positions) Then the same force calculations are done.
       nlist = 0
       do 200 i = 1,nmol-1
         jbeg = point(i)
         jend = point(i+1) - 1
c check that atom i has neighbors
         if (jbeg.le.jend) then
           rxi = x(i,1)
           ryi = x(i,2)
           rzi = x(i,3)
           fxi = fm(i,1)
           fyi = fm(i,2)
           fzi = fm(i,3)
           do 199 jnab = jbeg,jend
             j = list(jnab)
             nlist = nlist + 1
             rxij = rxi - x(j,1)
             ryij = ryi - x(j,2)
             rzij = rzi - x(j,3)
             rxij = rxij - dnint(rxij*vli)*vls
             ryij = ryij - dnint(ryij*vli)*vls
             rzij = rzij - dnint(rzij*vli)*vls
             rabs = dsqrt(rxij*rxij + ryij*ryij + rzij*rzij)
             rabi = 1/rabs
             if((mod(irep,lnay).eq.0).and.(mod(iloc,nloc).eq.0))then
               call log(fnb,nij,nlist,rxij,ryij,rzij)
               call shell(rxij,ryij,rzij)
               call raddist(irep,lnay,nmol,sig,vl)
             end if
             if(rabs.lt.rcut) then
               sr2 = rabi**2
               sr6 = sr2*sr2*sr2
               vij = 4*sr6*(sr6 - 1.)
               wij = 48*sr6*(sr6 -0.5)

```

```

           ve = ve + vij
           wu = wu + wij
           fij = wij*rabi**2
           fxij = rxij*fij
           fyij = ryij*fij
           fzij = rzij*fij
           fxi = fxi + fxij
           fyi = fyi + fyij
           fzi = fzi + fzij
           fm(j,1) = fm(j,1) - fxij
           fm(j,2) = fm(j,2) - fyij
           fm(j,3) = fm(j,3) - fzij
         end if
       199 continue
       fm(i,1) = fxi
       fm(i,2) = fyi
       fm(i,3) = fzi
     end if
   200 continue
   end if
   wu = wu/3.
   nlist = 0
c Calculate number of neighboring particles
c -----
   if((mod(irep,lnay).eq.0).and.(mod(iloc,nloc).eq.0)) then
     do 987 i = 1,nmol-1
       do 988 jnab = point(i),point(i+1)-1
         nlist = nlist + 1
         j = list(jnab)
         nnb(i) = nnb(i) + nij(nlist)
         nnb(j) = nnb(j) + nij(nlist)
       988 continue
     987 continue
     do 986 i = 1,nmol
       write(99,178) nnb(i)
     986 continue
   end if
   178 format(i5)
   return
   end

```

2.7 Subroutine SAVE

Saves the current configuration for future checking. It is called in subroutine FORCE.

```

c*****
c
c      subroutine save(fm,nmol,x)
c
c*****
c      implicit none
c      integer nmax
c      parameter (nmax=865)
c      integer i,nmol
c      double precision rxo,ryo,rzo
c      double precision fm(nmax,3),x(nmax,3)
c      common /block1/rxo(nmax),ryo(nmax),rzo(nmax)
c      do 100 i = 1,nmol
c        rxo(i)=x(i,1)
c        ryo(i)=x(i,2)
c        rzo(i)=x(i,3)
     100 continue
   return
   end

```

2.8 Subroutine LOG

LOG is called by FORCE to update the number of atoms in the neighbor list.

```

c*****
c
c      subroutine log(fnb,nij,nlist,rxlum,rylum,rzlum)
c
c*****
c      implicit none
c      integer nmax
c      parameter (nmax=865)
c      integer nlist,nij(nmax*(nmax-1)/2)
c      double precision fnb,rij,rxlum,rylum,rzlum
c
c      rij = (rxlum*rxlum + rylum*rylum + rzlum*rzlum)**0.5
c      nij(nlist) = idint(dsign(1.0D0,fnb-rij)+1.1)/2
c
c      return
c      end

```

2.9 Subroutine SHELL

```

c*****
c
c      subroutine shell(rxij,ryij,rzij)
c
c*****
c      implicit none

```

```

integer nrdels,nshell,ngofr
double precision gr,rdel,rij,rxij,ryij,rzij

common /rdf/ gr(500),rdel
common /rdf2/ nrdels,ngofr(500)

rij = (rxij*rxij + ryij*ryij + rzij*rzij)**0.5
nshell = rij/rdel + 0.5
ngofr(nshell) = ngofr(nshell) + 1

return
end

```

2.10 Subroutine CORRECTOR

After intermolecular forces are calculated, subroutine CORRECTOR is called to correct positions, velocities and higher order derivatives of the positions. These values are then used to update the configuration of molecules.

```

c*****
c
c      subroutine corrector(dts,fm,nmol,v,vls,x,iloc,irep)
c*****
c      calculate corrected values after force evaluation.
implicit none
integer nmax
parameter(nmax=865)
integer i,nmol,iloc,irep
double precision dts,v(nmax,3),x(nmax,3)
double precision accx,accy,accz,alp0,alp1,alp2,alp3,alp4,
+      axi,ayi,azi,bx,by,bz,c1,c2,c3,c4,corrxx,
+      corry,corrz,cx,cy,cz,cb,cc,cex,cv,vls,
+      vs1,vs2,vs3
double precision fm(nmax,3)
common /gear/ accx(nmax),accy(nmax),accz(nmax),
+      bx(nmax),by(nmax),bz(nmax),
+      cx(nmax),cy(nmax),cz(nmax)
common /correct/ alp0,alp1,alp2,alp3,alp4
c Coefficients of corrector step
c1 = dts
c2 = c1*dts/2.
c3 = c2*dts/3.
c4 = c3*dts/4.

cex = alp0*c2
cv = alp1*c2/c1
cb = alp3*c2/c3
cc = alp4*c2/c4
do 200 i = 1,nmol
c Accelerations calculated from Newton' s second law.
axi = fm(i,1)
ayi = fm(i,2)
azi = fm(i,3)
c Corrector factor for Gear predictor - corrector algorithm.
corrxx = axi - accx(i)
corry = ayi - accy(i)
corrz = azi - accz(i)
c Corrected values for positions.
x(i,1) = x(i,1) + cex*corrxx
x(i,2) = x(i,2) + cex*corry
x(i,3) = x(i,3) + cex*corrz
c Corrected values for velocities.
v(i,1) = v(i,1) + cv*corrxx
v(i,2) = v(i,2) + cv*corry
v(i,3) = v(i,3) + cv*corrz
c Total translation.
vs1=vs1+v(i,1)
vs2=vs2+v(i,2)
vs3=vs3+v(i,3)
accx(i) = axi
accy(i) = ayi
accz(i) = azi
c Corrected values for the third derivatives of the positions.
bx(i) = bx(i) + cb*corrxx
by(i) = by(i) + cb*corry
bz(i) = bz(i) + cb*corrz
c Corrected values for the fourth derivatives of the positions.
cx(i) = cx(i) + cc*corrxx
cy(i) = cy(i) + cc*corry
cz(i) = cz(i) + cc*corrz
call periodic(x(i,1),x(i,2),x(i,3),vls)
200 continue
c Total translation is gradually removed
do 280 i = 1,nmol
v(i,1)=v(i,1)-vs1/nmol*0.01
v(i,2)=v(i,2)-vs2/nmol*0.01
v(i,3)=v(i,3)-vs3/nmol*0.01
280 continue
return
end

```

2.11 Subroutine PERIODIC

This subroutine applies periodic boundary conditions. It is called by both subroutines STEP and CORRECTOR. Periodic boundary conditions are applied under the following conditions:

1. If $x(i,j) < 0$ then $x(i,j) = x(i,j) + vl$
2. If $x(i,j) > vl$ then $x(i,j) = x(i,j) - vl$

```

c*****
c
c      subroutine periodic(x,y,z,vls)
c*****
c      Apply pbc to particle in cubic box with the origin
c      of the coordinate system being at (0,0,0). The length
c      of the boundary is "vls" in dimensionless units.
implicit none
double precision vls,x,y,z
if(x.lt.0.0) then
x = x + vls
elseif(x.gt.vls) then
x = x - vls
endif
if(y.lt.0.0) then
y = y + vls
elseif(y.gt.vls) then
y = y - vls
endif
if(z.lt.0.0) then
z = z + vls
elseif(z.gt.vls) then
z = z - vls
endif
return
end

```

2.12 Subroutine AVG

AVG is used to calculate average potential and kinetic energy. It called in the main program after local loop is completed.

```

c*****
c
c      subroutine avg(enesum,nmol,velsq,vsqsum,v,ve,virsum,wu)
c*****
c      Average potential and kinetic energy are calculated by
c      subroutine AVG
implicit none
integer nmax
parameter(nmax=865)
integer i,nmol
double precision enesum,ve,velsq,vsqsum,virsum,wu
double precision v(nmax,3)
velsq = 0.
do 109 i = 1,nmol
velsq = velsq + v(i,1)**2. + v(i,2)**2. + v(i,3)**2.
109 continue
enesum = enesum + ve
virsum = virsum + wu
vsqsum = vsqsum + velsq
return
end

```

2.13 Subroutine OUTPUT

Positions, velocities, energies and average properties are output by this subroutine. It is called in the main program.

```

c*****
c
c      subroutine output(bk,enesum,eps,irep,kavg,lnay,lpo,lvo,
+      nloc,nmol,plr,sig,time,ulr,velsq,
+      vsqsum,v,ve,virsum,vls,wm,wu,x)
c*****
implicit none
integer nmax
parameter(nmax=865)
integer i,irep,j,kavg,nloc,nmol,lnay
integer nrdels,ngofr
double precision v(nmax,3),x(nmax,3)
double precision bk,denom,ek,ekavg,enesum,epot,epavg,eps,
+      etot,plr,preavg,press,sig,tmp,time,
+      tmpavg,ulr,ve,velsq,virsum,vsqsum,
+      vls,wm,wu
double precision gr,rdel,radius
logical lpo,lvo

```

```

common /rdf/ gr(500),rdel
common /rdf2/ nrdels,ngofr(500)
c
instantaneous properties
ek = 0.5*vlsq
epot = ve/nmol + ulr
etot = ek + epot
tmp = 2.*ek/(3.*nmol)
press = (wu + tmp*nmol)/vls**3 - plr*nmol*(1./vls)**3.
c
average properties
denom = dfloat(irep/kavg)
ekavg = 0.5*vsqsum/denom
epavg = enesum/(denom*nmol) + ulr
tmpavg = 2.*ekavg/(3.*nmol)
preavg = (virsum/denom + tmpavg*nmol)/vls**3.
+
- plr*nmol*(1./vls)**3.
-----
c
OUTPUT POSITION (x) (13)
-----
c
if(lpo)then
do 510 i = 1,nmol
write(13,'(3F8.3)') (x(i,j)*sig*1.0D10,j=1,3)
510 continue
endif
c
OUTPUT VELOCITY (v) (14)
-----
c
if(lvo)then
do 550 i = 1,nmol
write(27,'(3F10.3)') (v(i,j)/(wm/eps)**0.5,j=1,3)
550 continue
endif
c
OUTPUT INSTANTANEOUS PROPERTIES (16)
-----
c
write(16,'(F8.3,2x,F8.3,2x,f10.3,2x,f10.3,2x,f12.7)')
+
+ time*sig*(wm/eps)**0.5*1.0D12,
+ tmp,ek,
+ epot,press
c
OUTPUT INSTANTANEOUS PROPERTIES (30)
-----
c
write(30,'(F8.3,2x,F8.3,2x,f10.3,2x,f10.3,2x,f12.7)')
+
+ time*sig*(wm/eps)**0.5*1.0D12,
+ tmpavg,ekavg,
+ epavg,preavg
if(mod(irep,lnay).eq.0) then
do 800 j = 1,nrdels
radius = rdel*dfloat(j)
write(88,133) radius,gr(j)
800 continue
end if
133 format(f8.3,2x,f8.3)
return
end

```

2.14 Subroutine TEMCON

Subroutine TEMCON corrects the velocities with temperature. It is called by the main program.

$$V(i, j) = V(i, j) \sqrt{\frac{T_{ins}}{T_{mp}}} \quad (24)$$

where

$$T_{ins} = \frac{bk.T_{ini}}{\epsilon} \quad (25)$$

This allows the system to equilibrate at the initial temperature.

```

c*****
c
subroutine temcon(irep,nmol,tins,v)
c
c*****
implicit none
integer nmax
parameter(nmax=865)
integer i,irep,j,nmol
double precision ek,st,tins,tmp,v(nmax,3)
open(33,file='tempcon.dat')
write(33,*) irep
CLOSE(33)
ek = 0.
do 110 i = 1,nmol
ek = ek+v(i,1)*v(i,1) + v(i,2)*v(i,2) + v(i,3)*v(i,3)
110 continue
tmp = ek/(3.*nmol)
st = dsqrt(tins/tmp)

do 120 i = 1,nmol
do 120 j = 1,3
v(i,j) = v(i,j)*st
120 continue
return
end

```

2.15 Subroutine PROF

Pressure and density profiles are calculated in this routine. It is called in the main program.

```

c*****
c
subroutine prof(bk,eps,fnb,irep,lpo,lvo,nmol,pi,rcut,
+
sig,time,tini,tins,virsum,vl,vls,wm)
c
c*****
implicit none
integer nmax,ndr
parameter(nmax=865,ndr=110)
integer i,j,im,irep,j,jm,k,kk,nmol,nr1
integer lsti(nmax*(nmax-1)/2),lstj(nmax*(nmax-1)/2),
+
nr(ndr)
double precision epm(nmax),fij(nmax*(nmax-1)/2),
+
epij(nmax*(nmax-1)/2),fsij(nmax*(nmax-1)/2),
+
fs(nmax),pr(ndr),prb(ndr),
+
ri(nmax*(nmax-1)/2),rj(nmax*(nmax-1)/2),
+
rijq(nmax*(nmax-1)/2),v(nmax,3),x(nmax,3)
double precision bk,drp,dv,ep,epml,eps,fnb,
+
pi,prl,r1,r2,r3,rabs,rcut,ri1,ri2,ri3,rij,
+
riq,rj1,rj2,rj3,rjq,rk,rk1,rk2,rp0,sbr6,
+
sig,sk,sl,sv1,sv2,time,tini,v0,
+
v1,v2,vd,vd2,vi,vl,vli,wm
double precision tins,virsum,vls
common /one/ lsti,lstj,v,x
logical lpo,lvo
do 912 kk = 1,nmol
if(x(kk,1).lt.0.or.x(kk,2).lt.0.or.x(kk,3).lt.0.) then
print *, irep,"hold it right here"
end if
912 continue
vli = 1/vls
rp0 = 3.5D-10/sig
drp = .1
-----
c
POTENTIAL ENERGY AND PRESSURE
-----
c
ep = 0.0
vi = 0.0
do 150 i = 1,nmol
fs(i) = 0.0
epm(i) = 0.0
150 continue
do 160 i = 1,ndr
pr(i) = 0.0
prb(i) = 0.0
nr(i) = 0
160 continue
do 210 ij = 1,(nmol-1)*nmol/2
im = lsti(ij)
jm = lstj(ij)
-----
c
i-j DISTANCE
-----
c
r11 = x(im,1) - vls*0.5
r12 = x(im,2) - vls*0.5
r13 = x(im,3) - vls*0.5
rj1 = x(jm,1) - vls*0.5
rj2 = x(jm,2) - vls*0.5
rj3 = x(jm,3) - vls*0.5
riq = r11*r11 + r12*r12 + r13*r13
ri(ij) = dsqrt(riq)
rjq = rj1*rj1 + rj2*rj2 + rj3*rj3
rj(jj) = dsqrt(rjq)
r1 = x(im,1) - x(jm,1)
r2 = x(im,2) - x(jm,2)
r3 = x(im,3) - x(jm,3)
rijq(ij) = r1*r1 + r2*r2 + r3*r3
rij = dsqrt(rijq(ij))
r1 = r1 - dnint(r1*vli)*vls
r2 = r2 - dnint(r2*vli)*vls
r3 = r3 - dnint(r3*vli)*vls
rabs = dsqrt(r1*r1 + r2*r2 + r3*r3)
-----
c
L-J POTENTIAL AND FORCE*rij
-----
c
sbr6 = (1/rabs)**6
sl = dsign(0.5D0,vls*0.5-rabs)+0.5
epij(ij) = 4*(sbr6 - 1.0)*sbr6
ep = ep + epij(ij)*sl
sbr6 = (1/rij)**6
sl = dsign(0.5D0,vls*0.5-rij)+0.5
fij(ij) = 48*(sbr6-0.5)*sbr6*sl
fsij(ij) = dabs(fij(ij)/rij)
210 continue
-----
c
PRESSURE
-----
c
do 220 k = 1,ndr
rk = rp0 + drp*(k*1.0-0.5)
do 230 ij = 1,(nmol - 1)*nmol/2
riq = ri(ij)*ri(ij)
rjq = rj(ij)*rj(ij)
v0 = (riq - rjq)/rijq(ij)
vd2 = dmax1(0.0D0,v0*v0+1.0-2*(riq+rjq-2*rk*rk)/rijq(ij))
vd = dsqrt(vd2)
v1 = v0 - vd
v2 = v0 + vd

```

```

sv1 = dsign(0.5D0,1.0 - dabs(v1)) + 0.5
sv2 = dsign(0.5D0,1.0 - dabs(v2)) + 0.5
pr1 = 0.5*fsij(ij)*vd*(sv1 + sv2)
pr(k) = pr(k)+(dsign(0.5D0,-pr1)+0.5)*pr1
prb(k) = prb(k) + (dsign(0.5D0, pr1)+0.5)*pr1
230 continue
220 continue
-----
c DENSITY PROFILE
-----
do 250 i = 1,nmol
r1 = x(i,1) - vls*0.5
r2 = x(i,2) - vls*0.5
r3 = x(i,3) - vls*0.5
rabs = dsqrt(r1*r1 + r2*r2 + r3*r3)
k = idint((rabs-rp0)/drp) + 1
nr1=(dsign(1D0,rabs-rp0)+1)*(dsign(1D0,rp0+ndr*drp-rabs)+1)/4
k = max(k,1)
k = min(k,ndr)
nr(k) = nr(k) + nr1
250 continue
do 260 i = 1,nmol-1
do 260 j = i+1,nmol
k = j-i-1
ij = i+k*nmol - (1+k)*k/2b
fs(i) = fs(i) + fsij(ij)
fs(j) = fs(j) + fsij(ij)
260 continue
-----
c OUTPUT PRESSURE AND NEAREST NEIGHBOR RESULTS
-----
do 310 k = 1,ndr
rk = (rp0+drp*(k*1.0-0.5))*sig
sk = 4*pi*rk*rk
rk1 = (rp0+drp*(k-1))*sig
rk2 = rk1 + drp*sig
dv = 4*pi*(rk2**3-rk1**3)/3
write(15,'(3D12.3,2x,f10.3)') rk,pr(k)*eps/(sk*rk),
+ prb(k)*eps/(sk*rk),
+ nr(k)*wm/dv
310 continue
-----
c OUTPUT INDIVIDUAL PE and KE/atom
-----
do 520 ij = 1,(nmol-1)*nmol/2
im = lsti(ij)
jm = lstj(ij)
epml = epij(ij)*0.5
epm(im) = epm(im) + epml
epm(jm) = epm(jm) + epml
520 continue
return
end

```

3 DATA REDUCTION

Once thermal equilibrium had been reached at a desired temperature, the density profile throughout the drop could be calculated by determining the number of atoms, $N(r)$, in differential spherical shells of equal width, $\Delta r = 0.1\sigma$, through the following equation

$$\rho(r) = \langle N(r) \rangle / v(r) \quad (26)$$

where $\langle \rangle$ denotes an ensemble average taken over the duration of the simulation for which thermal equilibrium exists. The shell volume $v(r)$, (\AA^3) is simply represented by:

$$v(r) = \frac{4\pi\Delta r}{3} (3r^2 + \Delta r^2/4) \quad (27)$$

where r denotes the midpoint of the shell as measured from the center of the drop. The center of the drop coincided with the center of the simulation box because the center of mass of the drop was recalculated at each time step to compensate for any bulk motion of the drop. A regression analysis was then performed (Thompson *et al.*, 1984) to fit the density data with the commonly used hyperbolic tangent function such that

$$\rho(r) = \frac{1}{2}(\rho_\ell + \rho_v) - \frac{1}{2}(\rho_\ell - \rho_v) \tanh\left[\frac{2(r-r_0)}{d_s}\right] \quad (28)$$

where ρ_ℓ , (kg/m^3) and ρ_v , (kg/m^3) are the local liquid density at the center of the drop and the local vapor density at the boundary of the container, respectively and r_0 (\AA) is a coefficient designating estimate of the drop radius. The parameter d_s is a measure of the thickness of the surface layer about the equimolar plane at $r = r_0$.

4 NOMENCLATURE

accx,accy,accz = acceleration of atom (second derivative)
ai0 = sum of all the rotational displacements squared
alam = parameter for subroutine "lambda"
alp0,alp1,alp2,alp3,alp4 = constants of the gear-predictor algorithm
an = Avogadro's number (6.022e23 1/mole)
ang = random angle (similar to longitude of the earth)
axi,ayi,azi = forces on the particle (used in "corrector" subroutine)
base = initial distance from the particles on the outside of the crystal structure to the periodic boundary
bc = random magnitude
bk = Stefan Boltzman constant (1.381e-23 J/K)
bs = sqrt(1-bc^2)
bx,by,bz = third derivative of atom position
c1,c2,c3,c4 = dimensionless time parameters
coef = scaling coefficient
corrc,corry,corrz = difference between predicted acceleration and that given by the evaluated force in "force"
cossum = parameter for subroutine "lambda"
cx,cy,cz = fourth derivative of atom position
cb = correction parameters in "corrector" subroutine (alpha2)
cc = correction parameters in "corrector" subroutine (alpha3)
cex = correction parameters in "corrector" subroutine (alpha0)
cv = correction parameters in "corrector" subroutine (alpha1)
denom = averaging parameter
densy = density
dispmx = displacement used to determine whether neighbor needs to be updated
drp = size of the radial grid
dt = time step (sec)
dt2 = half of the dimensionless time step
dts = dimensionless time step
dtsq = 0.5*(dts^2)
dv = volume of the region ("prof")
ek = average kinetic energy per atom per unit mass
ekavg = average kinetic energy
enesum = average potential energy
ep = average potential energy per atom per unit mass
epavg = average potential energy
epij = potential energy contribution from interaction pair i,j
epot = average potential energy per atom per unit mass
epm = total potential energy
epml = half of epij
eps = minimum energy (8.27e-21 J)
etot = average total energy
ffc = face centering parameter: used to define a box within the control box used to encompass only the liquid drop
fij = force that atom i exerts on atom j
fm = forces on the particle
fnb = neighbor counting parameter
fofv = parameter in subroutine "veldist"
fs = total force exerted on atom i per unit distance
fsij = force that atom i exerts on atom j per unit distance
fxi,fyi,fzi = total forces on the particle
fxij,fyij,fzij = force that particle "i" exerts on "j"
gr = radial distribution function
hh = parameter in subroutine "veldist"
hinst = parameter in subroutine "veldist"
i = counter
ij = counter
iloc = counter for the execution of the local loop
im = atom labeling parameter

inte = frequency that PE & KE are output
 intlg = frequency that # of neighbours is output
 intp = frequency that the position vectors are output
 inttc = frequency of temperature control
 intv = frequency that the velocity is output
 irep = counter for the execution of the general loop
 ixs = parameter used to update position of liquid drop
 j = counter
 jbeg = counting parameter for neighbor atoms within "rlist"
 jend = counting parameter for neighbor atoms within "rlist"
 jh,jl,jlh,jll = parameters used in function "randf"
 jm = atom labeling parameter
 jnab = counter in "force" subroutine
 k = counter
 = parameter that places the atom in the correct radial increment or interval ("prof")
 kk = counter
 kavg = how often averaging of properties is done
 klatx,klaty,klatz = parameter for subroutine "lambda"
 kpnt = how often data is printed out
 list = list of all neighbors
 lnay = how often neighbor routine is called
 lpo = controls position vector output
 lsti = listvectors for interactions
 lstj = listvectors for interactions
 lvo = controls velocity vector output
 maxnab = limiting number for the neighbor list
 ndr = number of radial intervals ("prof")
 nij = adding parameter for interactive neighbor atoms
 nlist = number of atoms within the "rlist" radius
 nloc = # times local loop is executed
 nmax = number of particles in the simulation
 nmol = number of atoms
 nnb = neighbors
 nr = number of atoms in region
 nr1 = parameter which decides whether an atom is used in the calculation of the density profile
 nrrels = counting parameter
 nrndm = random seed
 nrep = # times general loop is executed
 nshell = number of shells
 ntc = # of temperature controls imposed
 nuni = unit cells in crystal
 nvdels = parameter in subroutine "veldist"
 nvols = unit cells in volume
 nxs = parameter used in updating position of liquid drop
 o1,o2,o3 = angular momentum (x,y,z)
 offset = parameter in subroutine "veldist"
 origins = parameter in subroutine "raddist"
 param = parameter for subroutine "lambda"
 pi = 3.1415927
 pi4d = parameter for subroutine "lambda"
 plr = long range pressure correction
 point = related to list (neighbor counting parameter)
 pr = pressure
 pr1 = parameter used to construct the pressure distribution
 prb = pressure distribution
 preavg = average pressure
 press = pressure
 r1,r2,r3 = position of particle minus half the box length ("init")
 = distance between atom i and j ("prof")
 = positions atom i w.r.t center of control volume ("prof")
 ra2 = sum of rotational displacements squared per atom
 rabi = reciprocal absolute distance between two atoms
 rabs = absolute distance between two atoms
 = absolute position w.r.t center of control volume ("prof")
 radius = radial distance between two atoms
 randf = random number generator: "FUNCTION randf"
 rcut = radial distance in which forces are considered
 rdel = width of the spherical shell used in "veldist"
 ri,rj = absolute position of i,j molecules
 ri1,ri2,ri3 = position of i molecule
 rj1,rj2,rj3 = position of j molecule
 rij = absolute distance between atoms i and j
 rijq = absolute distance between atoms i and j squared
 riq,rjq = absolute position of i,j molecules squared
 rk = point on grid midway between rk1 and rk2
 rk1,rk2 = first grid point before and after position "rk"
 rknt = counter
 rlist = distance larger than "rcut" including atoms that are within "rcut" before the next timestep
 rp0 = radial position of the first grid
 rxlum,rylum,rzlum = distance between two atoms
 rxi,ryi,rzi = position vectors
 rxij,ryij,rzij = distance between two atoms in x,y, and z
 rxo,ryo,rzo = dummy variable used in updating neighbor list
 sbr6 = rabs^(-6)
 scl = length of control box for liquid drop
 sig = collision diameter (potential separation distance)
 sinsum = parameter for subroutine "lambda"
 sk = volume at grid position
 sl = adding parameter for interactive neighbor atoms
 sr2 = rabs^(-2)
 sr6 = rabs^(-6)
 st = temperature control scalar
 sv1,sv2 = parameter used to construct the pressure distribution
 time = nondimensional time
 tini = initial temperature (K)
 tins = nondimensional temperature
 tmp = nondimensional absolute temperature
 tmpavg = average temperature
 ul = crystal length
 ulr = long range potential energy correction
 uls = nondimensional crystal length
 upcnt = tells the number of times neighbor list is updated
 update = logical
 v = velocity of particle
 v0 = parameter used in calculating pressure
 v1,v2,v3 = total translation
 v1,v2 = parameter used to construct the pressure distribution
 vd = parameter used to construct the pressure distribution
 vd2 = parameter used to construct the pressure distribution
 vdel = parameter in subroutine "veldist"
 ve = nondimensional velocity
 velsq = used to average kinetic energy
 vi = virial
 vij = nondimensional Lennard-Jones potential
 virsum = used to average virial
 vl = length of the simulation box (m)
 vli = reciprocal nondimensional length of box
 vls = nondimensional length of box
 vmax = parameter in subroutine "veldist"
 volshl = volume of shell
 vr1,vr2,vr3 = total rotation
 vs1,vs2,vs3 = velocity of the system
 vsqsum = sum of kinetic energy over time
 wj = force that atom i exerts on atom j (nondimensional)
 wm = mass of atom
 wu = used in force routine
 x,y,z = coordinate of position vector of particle
 xi = coordinate of position vector of particle
 xs1,xs2,xs3 = distance that the liquid drop is off center with respect to the center of the control volume

5 REFERENCES

- Allen MP, Tildesley DJ (1987) *Computer Simulation of Liquids*. Clarendon Press, New York, 17, 87
- Amini M, Fincham D (1990) Evaluation of Temperature in Molecular Dynamics Simulation. *Computer Phys Comm*, Vol. 56, 313-324
- Arnold A, Mauser N (1990) An Efficient Method of Bookkeeping Next Neighbors in Molecular Dynamics Simulations. *Computer Phys Comm*, Vol. 59, 267-275
- Bhansali AP, Bayazitoglu Y (1996), Molecular Dynamic Simulation of a Liquid Metal Using Oscillatory Pair Potential. *ICHMT Symposium Proceedings on Molecular and Microscale Heat Transfer in Material Processing and other Applications*, Vol 1, 90-104
- Bhansali AP, Bayazitoglu Y, Maruyama S (1996), Molecular Dynamic Simulation of an Evaporating Sodium Droplet. *Proceedings of 1996 ASME Heat Transfer Conference*, Vol 324, 137-147
- Bhansali AP, Bayazitoglu Y, Maruyama S (1999), Molecular Dynamic Simulation of an Evaporating Sodium Droplet, *Revue General de Thermique, Int. J. Therm. Sci.*, Vol. 38, 66-74
- Chialvo AA, Debenedetti PB (1991) On the Performance of an Automated Verlet Neighbor List Algorithm for

- Large Systems on a Vector Processor. *Computer Phys Comm*, Vol. 64, 15-18
8. Chokappa DK, Clancy P (1988) The Influence of an Interface in the Promotion of Melting, *Molecular Physics*, Vol. 65, pp.97-107
 9. Haile JM (1994) *Molecular Dynamics Simulation*. John Wiley & Sons, Inc., New York, 147-176, 231
 10. Hill PG, Witting H, Demetri EP (1963) Condensation of Metal Vapors During Rapid Expansion. *J Heat Transfer*, 303-317
 11. Kaltz T, Little J, Wong B, Micci M, Long LN (1994) Supercritical Droplet Evaporation Modeled Using Molecular Dynamics on Parallel Processors. *Euromech Colloquium 324, The Combustion of Drops, Sprays, and Aerosols*, Marseilles, France, 1-9
 12. Kotake S (1994) Future Aspects of Molecular Heat and Mass Transfer Studies. *Thermal Sci Eng*, Vol. 2, No. 1, 12-20
 13. Kotake S, Aoki (1996) Atomic and Molecular Clusters and Their Film Condensation. *ICHMT Symposium Proceedings on Molecular and Microscale Heat Transfer in Material Processing and other Applications*, Vol 1, 118-129
 14. Levesque D, Weis JJ, Reattol (1985), Pair Interaction form Structural Data for Dense Classical Liquids. *Physical Review Letters*, Vol 54, 451-454
 15. Lothe J, Pound GM (1969) Statistical Mechanics of Nucleation. *Nucleation* (Zettlemoyer AC, ed.), Marcel Dekker, Inc, New York, 112
 16. Long LN, Micci MM, Wong BC (1996) Molecular Dynamics Simulations of Droplet Evaporation. *Computer Physics Communications*, Vol. 96, 167-172
 17. Mandell MJ, McTague JP, Rahman A (1976) Crystal Nucleation in a Three-dimensional Lennard-Jones System: A Molecular Dynamics Study. *J Chem Phys*, Vol. 64, No. 9, 3699-3702
 18. Maruyama S, Matsumoto S, Ogita A (1994) Surface Phenomena of Molecular Clusters by Molecular Dynamics Method. *Thermal Sci Eng*, Vol. 2, No. 1, 77-84
 19. Matsumoto M (1996) Molecular Dynamics of Fluid Phase Change. *ICHMT Symposium Proceedings on Molecular and Microscale Heat Transfer in Material Processing and other Applications*, Vol. 1, 200-206
 20. Matsumoto M, Yasuoka K, Kataoka Y (1994) Microscopic Features of Evaporation and Condensation at Liquid Surfaces: Molecular Dynamics Simulation. Vol. 2, No. 1, 1-6
 21. Michaels AS (1969) *Proceedings of Nucleation Phenomena Conference*, A Chemical Study Publications, Washington, DC
 22. Pickering S, Snook I (1997) Molecular Dynamics Study of the Crystallisation of Metastable Fluids. *Physica A*, Vol.240, 297-304
 23. Pound GM (1952) Liquid and Crystal Nucleations. *Ind Eng Chemistry*, Vol. 44, No. 6, 1278-1283
 24. Reiss H (1952) Theory of the Liquid Drop Model. *Ind Eng Chemistry*, Vol. 44, No. 6, 1284-1288
 25. Rey C, Gallego LJ, Iniguez MP, Alonso JA (1992) A Molecular Dynamics Study of the Evaporation of Small Argon Clusters. *Physica B*, Vol. 179, 273-277
 26. Rowley RL (1994) *Statistical Mechanics for Thermophysical Property Calculations*. PTR Prentice Hall, Englewood Cliffs, New Jersey, 41, 236
 27. Santikary P, Bartell SL (1997) Molecular Dynamics Study of a Crystalline Cluster Undergoing a Second-Order transition in Inchoate Model of Acetylene. *J. Phys. Chem*. Vol. 101, 1299-1304
 28. Swope WC, Andersen HC, Berens PH, Wilson KR 1982 A Computer Simulation Method for the Calculation of Equilibrium Constants for the Formation of Physical Clusters of Molecules: Application to Small Water Clusters. *Journal of Chemical Physics*, Vol. 76, 637-649
 29. Thompson SM, Gubbins KE, Walton JPRB, Chantry RAR, Rowlinson JS (1984) A Molecular Dynamics Study or Liquid Drops. *J Chem Phys*, Vol. 81, No. 1, 530-522.
 30. Tsuruta T, Tanaka K, Tamashima K, Masuoka T (1996), Condensation Coefficient and Interphase Mass Transfer. *ICHMT Symposium Proceedings on Molecular and Microscale Heat Transfer in Material Processing and other Applications*, 243-254
 31. Verlet L (1967) Computer Experiments on Classical Fluids. I. Thermodynamic Properties of Lennard-Jones Molecules. *Physical Rev*, Vol. 159, 98-103
 32. Yasuoka K, Matsumoto M, Kataoka Y (1994) Evaporation and Condensation at a Liquid Surface. I. Argon. *J Chem Phys*, Vol. 101, No. 9, 7904-7911