NON-EQUILIBRIUM MOLECULAR DYNAMICS OF ELECTROMIGRATION IN ALUMINUM AND ITS ALLOYS

FATİH GÜRÇAĞ ŞEN

SEPTEMBER 2006

NON-EQUILIBRIUM MOLECULAR DYNAMICS OF ELECTROMIGRATION IN ALUMINUM AND ITS ALLOYS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

FATİH GÜRÇAĞ ŞEN

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
METALLURGICAL AND MATERIALS ENGINEERING

SEPTEMBER 2006

Approval of the Graduate School of Natural and Applied Sciences

Prof. Dr. Canan ÖZGEN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. Tayfur ÖZTÜRK
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Assoc. Prof. Dr. M. Kadri AYDINOL
Supervisor

Examining Committee Members

Prof. Dr. Şakir BOR (METU, METE) _____

Assoc. Prof. Dr. M. Kadri AYDINOL (METU, METE) _____

Prof. Dr. İshak KARAKAYA (METU, METE) _____

Assoc. Prof. Dr. Hakan GÜR (METU, METE) _____

Dr. Kaan PEHLİVANOĞLU (TÜBİTAK, SAGE) _____

I hearby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required, I have fully cited and referenced all material and results that are not original to this work.

Name Lastname : FATİH GÜRÇAĞ ŞEN

Signature :

Abstract

NON-EQUILIBRIUM MOLECULAR DYNAMICS OF ELECTROMIGRATION IN ALUMINUM AND ITS ALLOYS

ŞEN, FATİH GÜRÇAĞ
M.Sc., Department of Metallurgical & Materials Engineering
Supervisor: Assoc. Prof. Dr. Mehmet Kadri AYDINOL

September 2006, 99 pages

With constant miniaturization of integrated circuits, the current densities experienced in interconnects in electronic circuits has been multiplied. Aluminum, which is widely used as an interconnect material, has fast diffusion kinetics under low temperatures. Unfortunately, the combination of high current density and fast diffusion at low temperatures causes the circuit to fail by electromigration (EM), which is the mass transport of atoms due to the momentum transfer between conducting electrons and diffusing atoms. In the present study, the effect of alloying elements in aluminum on the diffusion behaviour is investigated using a non equilibrium molecular dynamics method (NEMD) under the effect of electromigration wind force. The electromigration force was computed by the use of a pseudopotential method in which the force depends on the imperfections on the lattice. 1.125 at% of various elements, namely Cu, Mg, Mn, Sn and Ti were added into aluminum. The electromigration force was then calculated on the alloying elements and the surrounding aluminum atoms and these forces incorporated into molecular dynamics using the non-equilibrium formalism. The jump frequencies of aluminum in these systems were then computed. Cu, Mn and Sn impurities were found to be very effective in lowering the kinetics of the

diffusion under electromigration conditions. Cu was known experimentally to have such an effect on aluminum for several years, but the Mn and Sn elements are shown here for the first time that they can have a similar effect.

Keywords: Electromigration, Diffusion, Non-Equilibrium Molecular Dynamics, Aluminum Interconnect.

Öz

ALÜMİNYUM VE ALAŞIMLARINDA ELEKTROGÖÇÜN MOLEKÜLER DİNAMİK SİMÜLASYONU

ŞEN, FATİH GÜRÇAĞ
Y. Lisans, Metalurji ve Malzeme Mühendisliği Bölümü
Tez Yöneticisi: Doç. Dr. M. Kadri AYDINOL

Eylül 2006, 99 sayfa

Entegre devrelerin sürekli küçülmesiyle elektronik devre bağlantı elemanlarında oluşan akım yoğunluğu katlanarak artmaktadır. Bu tür devrelerde geniş ölçüde kullanılan Alüminyum metali, düşük sıcaklıkta hızlı yayınım devinimine sahiptir. Aynı anda hem yüksek akım yoğunluğu, hem de hızlı yayınım devinimi, devrelerin elektrogöç nedeniyle hızlı bir şekilde bozulmasına neden olmaktadır. Elektrogöç, akım içinde ilerleyen elektronların momentumlarını atomlara aktarmalarıyla oluşan kütle hareketidir. Bu çalışmada, alüminyuma eklenen alaşım elementlerinin yayınım davranışına etkisi, elektrogöç kuvveti altında, denge dışı moleküler dinamik yöntemiyle incelenmiştir. Elektrogöç kuvveti, kuvvetin kafes içindeki hatalara bağlı olduğu psödopotansiyel metodu ile hesaplanmıştır. Çalışmada, Cu, Mg, Mn, Sn ve Ti elementleri, %1,125 atom oranında alüminyuma eklenmiştir. Elektrogöç kuvveti, eklenen alaşım elementleri ve bu elementleri çevreleyen alüminyum atomları üzerinde hesaplanarak denge dışı durumu ile moleküler dinamik yöntemine dahil edilmiştir. Daha sonra alüminyum atomlarının atlama frekansı hesap edilmiştir. Elektrogöç koşulları altında yayınım devinimi azaltmada Cu, Mn ve Sn katkılarının çok etkili olduğu görülmüştür. Bakırın, alüminyum içinde böylesi bir etkiye sebep olduğu deneysel çalışmalar sonu-

cunda uzun zamandır bilinmektedir ancak Mn ve Sn elementlerinin de buna benzer etkileri olduğu ilk defa gösterilmiştir.

Anahtar Kelimeler: Elektrogöç, Yayınım, Moleküler Dinamik, Alüminyum bağlantı elemanı

To my family

Acknowledgements

I would like to express my sincere gratitude to my supervisor Kadri Aydınol for his valuable guidance and support during the study of this thesis. I also thank to him for his endless patience to me.

I deeply thank my friends, Tufan Güngören and Alper Kınacı for their valuable suggestions and support during the period of this study.

I specially thank to my friends Evren Tan, Güher Kotan, Melih Topçuoğlu, Barış Okatan, Hasan Akyıldız, Betül Akköprü, Ziya Esen, Ali Erdem Eken, Doruk Doğu, Caner Şimşir, Kemal Davut, Gül Çevik, Selen Gürbüz for their valuable support during the period of writing this thesis.

I am grateful to my family, Azize, Şadi and Ayça Şen for their encouragement and constant support during the period of this study.

The numerical calculations reported in this study were carried out at the ULAK-BIM High Performance Computing Center at the Turkish Scientific and Technical Research Council (TUBITAK). I specially thank to the member of this facility Onur Temizsoylu for his constant help during the usage of this super computing facility.

Table of Contents

Chapter 1

Introduction

Computer simulations play a very important role in materials science today. Computational materials science acts as a bridge between experimental and theoretical approaches. Computer simulations are often used both to solve theoretical models beyond certain approximations and to provide a hint to experimentalists for further investigations. Understanding the properties of materials in terms of their structure and the microscopic interactions between them can be established by them. This serves as a complement to conventional experiments. In addition to that, simulations can be utilized not only to understand and interpret the experiments, but also to study regions which are not accessible experimentally, or which would imply very expensive experiments. Investigation of material properties and materials design by using computers is cost effective, in which the cost of characterization, synthesis, processing and testing of materials are eliminated with the use of computer experiments.

The traditional simulation methods for many-body systems can be divided into two classes of stochastic and deterministic simulations, which are largely covered by the Monte Carlo (MC) method [1] and the molecular dynamics (MD) method [2, 3, 4], respectively. In addition to that there is a whole range of hybrid techniques which combine features from both. Monte Carlo simulations probe the configuration space by trial moves of particles. Within the so-called Metropolis [1] algorithm, the energy change between the following steps is used as a trigger to accept or reject the new configuration. Paths towards lower energy are always accepted, those to higher energy are accepted with a probability governed by Boltzmann statistics. In that way, properties of the system can be calculated by averaging over all Monte Carlo moves (where one move means that every degree of freedom is probed once on average) [5].

By contrast, in MD methods; Newton's equations of motion are integrated to move particles to new positions and to get new velocities at these new positions. The state of a dynamic system is given by its phase space coordinates at a certain time. In MD, a dynamic system moves and changes its phase space coordinates in time, evolving towards an internal equilibrium state of the whole system. The macroscopic observables of the system are then obtained by suitably defined averages over the phase space trajectory, at the internal equilibrium state of the whole system [6]. Molecular dynamics simulations allow us to calculate static properties as well as dynamic properties of the system including transport properties.

Transport properties in materials are important as they describe how a material relaxes back to equilibrium, following application of a mechanical or thermal perturbation [7] . Introducing such perturbations into the equations of motion, produce a time-dependent non-equilibrium distribution. At this point non-equilibrium molecular dynamics (NEMD) method arises, which is firstly announced by Hoover and Ashurst [8]. NEMD methods are used to calculate transport properties such as viscosity, self- and mutual diffusion coefficients, and thermal conductivity more precisely than equilibrium methods [9].

Electromigration is the mass transport of a metal due to the momentum transfer between conducting electrons and diffusing metal atoms. Electromigration has been known for 100 years, but it is concerned with the development of the integrated circuits. The thin films used as interconnects in integrated circuits have a thickness in the order of 300 nm. Although a relatively small voltage (3.5-3.6 V) passes from these interconnects, because of the small length over the applied voltage, very high electric field was developed and consequently because of the small cross-section, very high current density ($10^6 \mathrm{A/cm}^2$) resulted due to Ohm's Law. Furthermore, the conductors were made of pure aluminum, a material with a low melting temperature, which implies fast diffusion at low temperatures. Such very thin film contains small grains and thus many grain boundaries that are suitable for rapid diffusion. This combination of high current density and fast diffusion at low temperatures causes the circuit to fail due to vacancy diffusion induced void formation and growth especially at grain boundaries. Intensive research has been carried out to overcome this failure, for example, by adding few percentages of alloying elements like copper [10].

The electromigration was investigated theoretically in atomistic and phenomenological point of view in the literature. In the last decade several computer simulations of electromigration in metal lines have been reported from the macroscopic point of

view [11, 12, 13]. However, as circuit integration shrinks to smaller dimensions, influences of crystal orientation and grain boundary structure on the lifetime must be taken into account.

From the physics point of view, Sorbello [14, 15, 16] has made a very comprehensive study about determining the driving force for electromigration for each individual atom in a lattice. Sorbello has studied the atomic configuration-dependent electromigration force including impurities and vacancies.

There are very limited studies in the literature involving atomistic simulations. Molecular dynamics study of electromigration has been carried out by Ohkubo et al [17]. In that study H type periodic boundaries for interconnects has been constructed and 2 dimensional molecular dynamics simulations were carried out for aluminum. In that study, the electromigration driving force was calculated using Cloud in Cell (CIC) method and the evolution of the void formation was reported. Shinzawa and Ohta [18] characterized the grain boundary diffusion for aluminum interconnects by molecular dynamics. The diffusion characteristics with respect to the grain boundary angle was reported. Maroudas and Gungor [19] studied the void evolution and failure in metallic thin films. In that study, plastic deformations in the vicinity of the voids are examined by the use of molecular dynamics.

In the development of an alloy with improved resistance to electromigration, we should somehow eliminate grain boundaries or slow down the diffusion process. Because of the stages in the processing of these interconnects, it is inevitable to have grain boundaries. However it is possible to alter the diffusion behavior with the addition of alloying elements. Then the answer to the following question should be given: Which alloying element would be effective in lowering the diffusion kinetics. In the present study, therefore the effect of alloying elements in aluminum to the bulk diffusion behavior is investigated using molecular dynamics method, under the effect of electromigration wind force.

Initially the MD method was reviewed in Chapter 2. The construction of a MD simulation is explained in detail, including how a MD simulation can be started, how the atomic interactions are handled, the integration methods used and how the properties of a material can be determined with the given algorithms. In Chapter 3, the derivation of potentials is reported by introducing the pseudopotential method, and how a pair potential can be obtained using this method. In Chapter 4, the pseudopotential approach to the electromigration due to Sorbello was given. In Chapter 5, the methodology followed is reported and the results obtained were given in Chapter 6.

Chapter 2

Molecular Dynamics

2.1 Classical Molecular Dynamics

2.1.1 History of Molecular Dynamics

The foundations of the Molecular Dynamics (MD) method has been laid by the development of Newtonian dynamics by Isaac Newton. He proposed that the same mechanical laws can explain the motions of all bodies, large and small, with suitable definitions of the forces operative. Thus the Newtonian laws applies to systems of any size, even to atoms. This concept has been modified by quantum mechanics. Then in classical statistical mechanics, Ludwig Boltzmann investigated the problem of correlating the detailed dynamic behavior of a system of atoms and molecules with the macroscopic experimentally measurable properties of the same system. These two scientific approaches of classical dynamics and classical statistical mechanics constitute the basis of the MD method. The trajectories and velocities of the particles corresponding to atoms (or molecules, or ions) are generated using Newtonian equations of motion. Classical statistical mechanical concepts are then used to obtain the correspondence of this system to a thermodynamic system.

The problem of studying the interaction of many atoms or molecules has only become possible, using the methods of MD and MC, with the development of powerful electronic computers after 1950's.

The first papers reporting a molecular dynamics simulation were written by Alder and Wainwright in 1957 [2, 3]. The purpose of the paper was to investigate the phase diagram of a hard sphere system, and in particular the solid and liquid regions. In a hard sphere system, particles interact via instantaneous collisions, and travel as free

particles between collisions.

Probably the first example of a molecular dynamics calculation with a continuous potential based on a finite difference time integration method was done by Gibson et al [20]. In that study, the calculation for a 500-atoms system was performed on an IBM 704, and took about a minute per time step. A successful attempt to solve the equations of motion for a set of Lennard-Jones particles was made by Rahman [4]. Since that time the properties of the Lennard-Jones model have been thoroughly investigated by Verlet [21, 22] in which *Verlet time integration algorithm* was used.[23]

After initial foundations on atomic systems, molecular dynamics simulation developed rapidly. Today molecular dynamics simulations are used in various research areas, such as liquids, defects, fracture, surfaces, friction, clusters, biomolecules, electronic properties and dynamics. [23]

2.1.2 Basics of Molecular Dynamics

In molecular dynamics, the laws of classical mechanics are followed, and most particularly Newton's law of equation of motion:

$$F_i = m_i a_i \tag{2.1.1}$$

for each atom i in a system constituted by N atoms. Here m_i is the atomic mass, $a_i = \frac{d^2 r_i}{dt^2}$ its acceleration, r_i its position and F_i the force acting upon it, due to the interactions with other atoms. Therefore in MD, given an initial set of positions and velocities, the subsequent time evolution of the system is determined. The macroscopic or thermodynamical properties are then calculated with the help of statistical mechanics.

MD simulation consists of three steps; initialization, equilibration and thermalization. In initialization part, initial positions and velocities are given to the atoms and initial system parameters of the simulation are set. Then, equations of motion are solved until the system is reached to thermal equilibrium in the equilibration part. After the thermal equilibrium, the thermodynamic properties are calculated in thermalization step.

In Figure 2.1.1 the algorithm of a basic molecular dynamics simulation has been shown, where in the following sections the parts of the molecular dynamics method will be covered in detail.

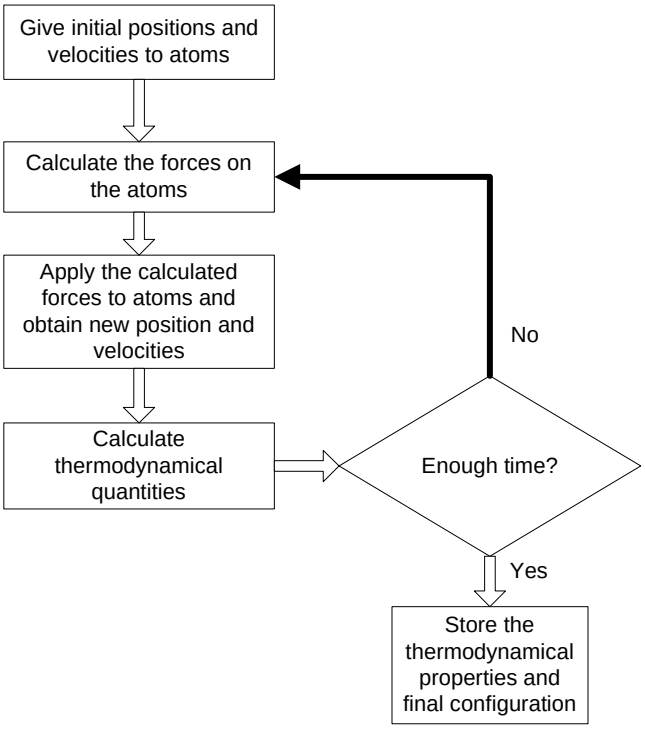

Figure 2.1.1: Basic molecular dynamics simulation algorithm.

Initialization

To start a simulation, the MD box should be defined and the set of positions and velocities should be assigned initially to the particles. There are two common ways of doing this:

i) Starting from scratch If the simulation is starting from scratch, a set of initial positions and velocities should be created.

Positions are usually defined on a lattice, assuming a certain crystal structure. This structure is typically the most stable one at 0 K with the given potential. Assigning positions to atoms should be carefully handled so that there should not be any overlap of atoms.

The initial velocities may be taken to be zero or can be assigned by taking them from a Maxwell distribution at a certain temperature T_{set}. When doing this, the system will have a small total linear momentum, corresponding to a translational motion of the whole system. Since this is somewhat inconvenient to have, it is common practice to subtract this component from the velocity of each particle in

order to operate in a zero total momentum condition. Initial temperature of the system is also given in the initialization step by adjusting the kinetic energy. The instantaneous temperature, $T(t)$ of the system is given by the relation 2.1.2

$$N_f k_b T(t) = \sum_{i=1}^{N} m_i v_i^2 \tag{2.1.2}$$

where N_f is the degrees of freedom ($N_f = 3N - 3$ for a system of N particles with zero total linear momentum), k_b is the Boltzmann constant and v_i is the velocity on atom i. The right hand side of Equation 2.1.2 is two times of the average kinetic energy. This equation comes from the equipartition principle [24]: an average energy of $k_b T/2$ per degree of freedom. The desired temperature, T_{set}, is obtained by scaling all velocities with a factor $(T_{set}/T(t))^{1/2}$.

Such an initial state will not of course correspond to an equilibrium condition. However, once the run is started, depending on the system size, equilibrium is usually reached within a time of the order of several hundred time steps.

ii) Continuing a simulation Another possibility is to take the initial positions and velocities to be the final positions and velocities of a previous MD run. This is in fact the most commonly used method in actual production. For instance, if one has to take measurements at different temperatures, the standard procedure is to set up a chain of runs, where the starting point at each temperature is taken to be the final point at the preceding temperature (lower if we are heating, higher if we are cooling).

Atomic Interactions

In order to estimate the force acting between atoms, it is necessary to take into account the elementary components of the atom. With the normal concept of an atom made up of a central nucleus and orbital electrons, when it interacts with an another atom, if there were no orbital electrons the force between two nuclei would be Coulombic. The forces holding together the nuclear components - neutrons, protons - are of a completely different nature and orders of magnitude stronger than any interatomic forces.

If atoms were considered as entities, there should be no forces between them at distances greater than that at which they touch. However, at separations comparable

with the atomic diameter, the Coulomb interactions of the outer electrons of atom 1 with those of atom 2 and with the nucleus of atom 2 (and vice versa) have a significant effect. The two atoms no longer view each other as entities and the interaction becomes a rather complicated many-body problem.

Treatment of this many-body problem for all types of atomic interactions, requires powerful computing systems. Fortunately, a number of approximations and averaging procedures under different physical situations enable us to obtain analytical expressions or numerical values for the interatomic forces which may be applied with varying degrees of confidence to atomic problems [25].

For most purposes the force between two atoms is expressed in terms of their potential energy of interaction, or interatomic potential. It depends to a first approximation on the separation r between the atoms; the relation between the force $F(r)$ and the potential $V(r)$ is

$$F(r) = -(\frac{\partial}{\partial r})V(r) \tag{2.1.3}$$

The potential energy of an atom is the work done in bringing all components of the atom from infinity to their equilibrium positions. The potential energy baseline then is that of a system with the nucleus already fully constituted. The work required to attach the atomic electrons to the nucleus under the force fields of the nucleus and of each other [25]. The potential energy $V(r)$, representing non-bonded interactions between atoms is traditionally split into 1-body, 2-body, 3- body,... terms;

$$V(r_N) = \sum_i u(r_i) + \sum_i \sum_{j>i} \varphi(r_i, r_j) + ... \tag{2.1.4}$$

where $r_N = (r_1; r_2; ...r_N)$ represents the complete set of 3N coordinates. The $u(r)$ term in Equation 2.1.4 represents an externally applied potential field or the effects of the container walls; it is usually dropped for fully periodic simulations of bulk systems [26]. Also it is usual to concentrate on the pair potential $\varphi(r_i, r_j)$ and neglect three body and higher order interactions. There is an extensive literature on the way these potentials are determined experimentally, or modelled theoretically [25]. In some simulations of complex fluids, it is sufficient to use the simplest models that faithfully represent the essential physics.

Integration of the Equations of Motion

The engine of a molecular dynamics program is its time integration algorithm, required to integrate the equation of motion of the interacting particles. The integrator is responsible for the accuracy of the simulation results. Sutmann [5] summarized the requirements of an integrator, such that an integrator should be accurate to obtain a true trajectory, stable in the sense that it conserves energy and that small perturbations do not lead to instabilities. An integrator should also be robust that it permits a large time step to be used in order to propagate the system efficiently through phase space.

Time integration algorithms are based on finite difference methods, where time is discretized on a finite grid, the time step Δt being the distance between consecutive points on the grid [23]. Knowing the positions and some of their time derivatives at time t, the integration scheme gives the same quantities at a later time $t + \Delta t$. By iterating the procedure, the time evolution of the system can be followed for long times. Generally integration algorithms are based on Taylor expansion of positions and velocities. Some of the common integrators used in molecular dynamics simulations are described in the following sections.

Verlet Algorithm Among integration algorithms for equations of motion, Verlet algorithm [21] is the most widely used. In that method the advancing positions of the atoms are calculated using the positions of atoms $r(t)$, accelerations of atoms $a(t)$, and the positions at the previous step $r(t - \Delta t)$. The equation governing this method is :

$$r(t + \Delta t) = 2r(t) - r(t - \Delta t) + \Delta t^2 a(t) \tag{2.1.5}$$

Accelerations in Equation 2.1.5 are calculated by using Equation 2.1.1. As can be seen from Equation 2.1.5, in that method velocities are not needed to compute the trajectory of atoms. However, since velocities are necessary to estimate the kinetic energy, velocities can be obtained by the relation;

$$v(t) = \frac{r(t + \Delta t) - r(t - \Delta t)}{2\Delta t} \tag{2.1.6}$$

It is seen from Equation 2.1.6 that the velocities at time t can only be calculated after the positions at time $t + \Delta t$ were obtained. In order to calculate positions

and velocities at the same time, Verlet algorithm has been modified [27] including 'leapfrog' [28] and 'velocity Verlet' [29] form. In velocity Verlet scheme, positions and velocities at $t + \Delta t$ are obtained from the same quantities at time t. The equations of this method are:

$$r(t + \Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 \tag{2.1.7}$$

$$v(t + \frac{\Delta t}{2}) = v(t) + \frac{1}{2}a(t)\Delta t \tag{2.1.8}$$

$$a(t + \Delta t) = -\frac{1}{m}\nabla V(r(t + \Delta t)) \tag{2.1.9}$$

$$v(t + \Delta t) = v(t + \frac{\Delta t}{2}) + \frac{1}{2}a(t + \Delta t)\Delta t \tag{2.1.10}$$

The error associated to the velocity - Verlet method is of order Δt^4 where, original Verlet has an error, order of Δt^2. Velocity Verlet form of the integrator resembles a three-value predictor-corrector algorithm where the position coefficient is zero [26].

Predictor-corrector Algorithm Predictor-corrector algorithms commonly used to integrate the equations of motion used in molecular dynamics are due to Gear [30, 31], and consists of three steps [23]:

1. *Predictor:* The positions and their time derivatives up to a certain order q are predicted at time $t + \Delta t$ using their current values by means of a Taylor expansion.

2. *Force evaluation:* The force is computed by Equation 2.1.3 using the predicted positions.

3. *Corrector:* The difference between the accelerations calculated from force evaluation and predicted accelerations is used to correct positions and their derivatives. All the corrections are proportional to the error signal, the coefficient of proportionality being a magic number determined to maximize the stability of the algorithm.

In the predictor part, the estimate of positions, velocities, accelerations, etc. at time $t + \Delta t$ is obtained by Taylor expansion at time t. For the fifth order Gear predictor-corrector method the equations are written as:

$$\begin{aligned} r^p(t+\Delta t) &= r(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2 + \frac{1}{6}b(t)\Delta t^3 + \frac{1}{24}c(t)\Delta t^4 \\ v^p(t+\Delta t) &= v(t) + a(t)\Delta t + \frac{1}{2}b(t)\Delta t^2 + \frac{1}{6}c(t)\Delta t^3 \\ a^p(t+\Delta t) &= a(t) + b(t)\Delta t + \frac{1}{2}c(t)\Delta t^2 \\ b^p(t+\Delta t) &= b(t) + c(t)\Delta t \end{aligned} \tag{2.1.11}$$

The superscript p designate the predicted values. r, v, a, denotes the positions, velocities and accelerations respectively and b and c are third and fourth derivatives of r. The predicted equations given above do not give the true trajectory of atoms since the equations of motion have not been introduced. The corrected accelerations, $a^c(t + \Delta t)$, are calculated using the new positions, r^p, from the forces at time $t + \Delta t$. These corrected accelerations are then compared with the predicted values to estimate the size of the error.

$$\Delta a(t + \Delta t) = a^c(t + \Delta t) - a^p(t + \Delta t) \tag{2.1.12}$$

Using this error positions, velocities, etc. are corrected.

$$\begin{aligned} r^c(t+\Delta t) &= r^p(t+\Delta t) + c_0\Delta a(t+\Delta t) \\ v^c(t+\Delta t) &= v^p(t+\Delta t) + c_1\Delta a(t+\Delta t) \\ a^c(t+\Delta t) &= a^p(t+\Delta t) + c_2\Delta a(t+\Delta t) \\ b^c(t+\Delta t) &= b^p(t+\Delta t) + c_3\Delta a(t+\Delta t) \\ c^c(t+\Delta t) &= c^p(t+\Delta t) + c_4\Delta a(t+\Delta t) \end{aligned} \tag{2.1.13}$$

where the superscript c donates the corrected values and c_0, c_1, c_2, c_3, c_4 are the parameters suggested by Gear [30, 31]. These parameters are given in Table 2.1.1.

Table 2.1.1: Gear corrector coefficients

Order	c_o	c_1	c_2	c_3	c_4	c_5
3	0	1	1			
4	1/6	5/6	1	1/3		
5	19/120	3/4	1	1/2	1/12	
6	3/20	251/360	1	11/18	1/6	1/60

Periodic Boundary Conditions

The atoms on the boundaries of the simulation cell have less neighboring atoms than the atoms located at the inside. This cause a surface effect on the simulation which is not desired for a bulk simulation. This surface effect problem can be overcome by implementing periodic boundary conditions [32]. In that method the cubic box is replicated throughout space to eliminate surfaces as shown in Figure 2.1.2 . In the simulation as an atom moves in the original box, its periodic image in each of neighboring boxes moves in exactly the same way. Therefore as an atom leaves the central box, one of its images will enter through the opposite face.

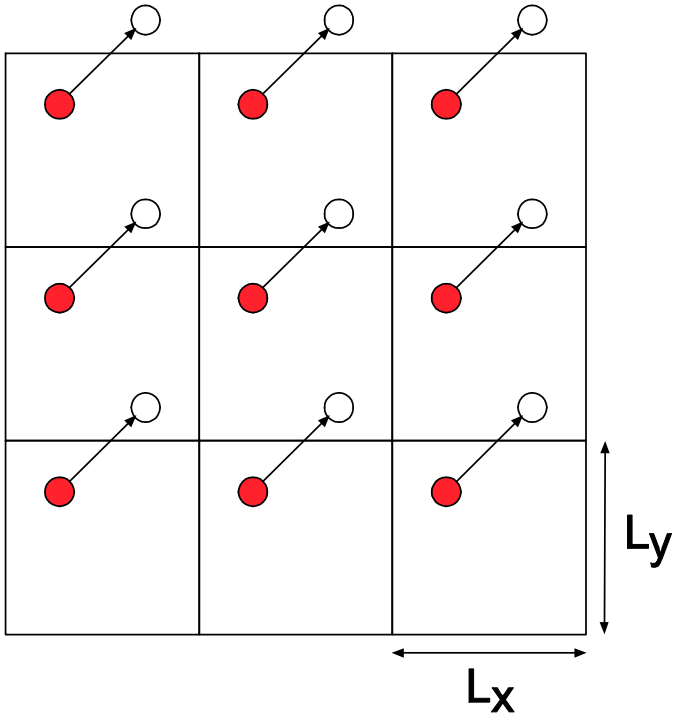

Figure 2.1.2: Periodic boundary conditions. As a particle moves out of the simulation box, an image particle moves in to replace it from the opposite side.

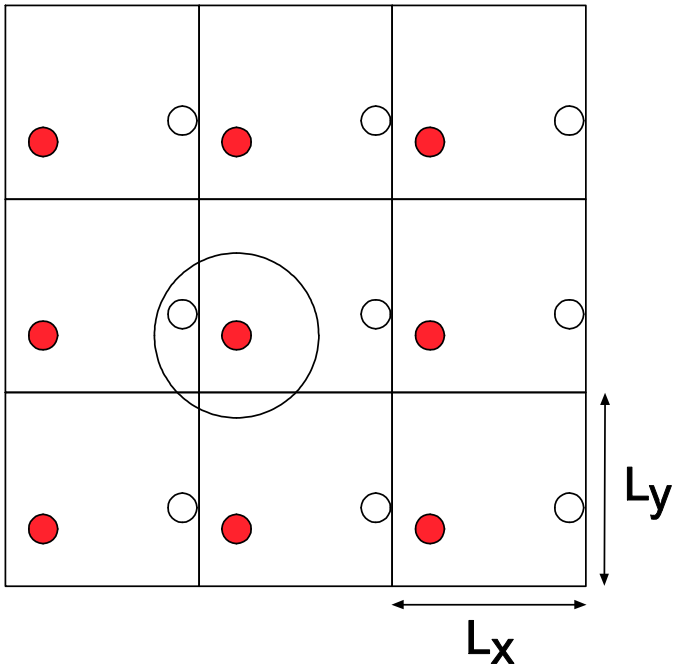

Figure 2.1.3: The minimum image convention in a two-dimensional system. In calculating particle interactions within cutoff range, both real and image neighbours are included.

The implementation of periodic boundary conditions for a cubic box with box length L, is given in Algorithm 1. When an atom leaves the box, by adding or subtracting the box length to the appropriate coordinate, an image of the atom is moved in to the box.

Algorithm 1 *Periodic boundary condition for a cubic box with box length L*

if $(rx_i < 0)$ *then* $rx_i = rx_i + L$
if $(rx_i > L)$ *then* $rx_i = rx_i - L$

Minimum Image Convention

In molecular dynamics, potential energy and forces acting on all atoms, for a system subjected to periodic boundary conditions, are evaluated by minimum image condition. In this method, the atomic interaction with the nearest image of all the particles in the simulation box is calculated, shown as circle in Figure 2.1.3.

In the minimum image convention, the calculation of the potential energy due to pairwise -additive interactions involves $\frac{1}{2}N(N-1)$ terms. This requires a lot of

computation time even for small systems. The largest contribution to the potential and forces come from the neighbors close to the atom of interest, and for short-range forces generally a spherical cutoff is applied. With a cutoff distance r_c, the pair potential $\varphi(r)$ is set to zero for $r \geq r_c$. The cutoff distance should be large enough to minimize the perturbation because of the use of cutoff. The cut off distance should be less than $\frac{L}{2}$ for consistency with the minimum image convention [26]. The implementation of minimum image convention is similar to periodic boundary conditions as can be seen in Figure 2.1.3. In the given Algorithm 2, *rxij* is the interatomic separation of atoms in interest with the cubic simulation box length L.

Algorithm 2 *Minimum image convention for a cubic box with box length L*

if $(rx_{ij} < -L/2)$ then $rx_{ij} = rxij + L$
if $(rx_{ij} > L/2)$ then $rx_{ij} = rxij - L$

Linked List

In the evaluation of forces and potential energy for an atom all of the minimum image distances are calculated and if the pairs are separated greater than a potential cutoff, the evaluation process is not carried out. If the system size increases towards 1000 atoms, this evaluation sequence becomes very time consuming. To avoid the needless calculation of pair separations that are so large, linked list algorithm can be used.

In this method at first, the simulation box, L, is divided into $M \times M \times M$ small cells of equal size. The dimensions of the cell, $l = \frac{L}{M}$ should be greater than the cutoff distance, r_c. An atom in a cell interacts with only the other atoms in the same cell and its 26 neighbor cells.

In the first part of the method, all of the atoms are sorted into their appropriate cells. Two arrays are created during the sorting process. The 'head-of-chain' array (Head) has one element for each cell, that contains the identification number of one of the atoms sorted into that cell. This number is used to address the element of a linked-list array (List), which contains the number of the next atom in that cell. In turn 'List' array element for that atom is the index of the next atom in the cell, and so on. If this sequence is followed, eventually it is reached an element of 'List' which is empty [26]. The illustration of this method can be seen in the Figure 2.1.4.

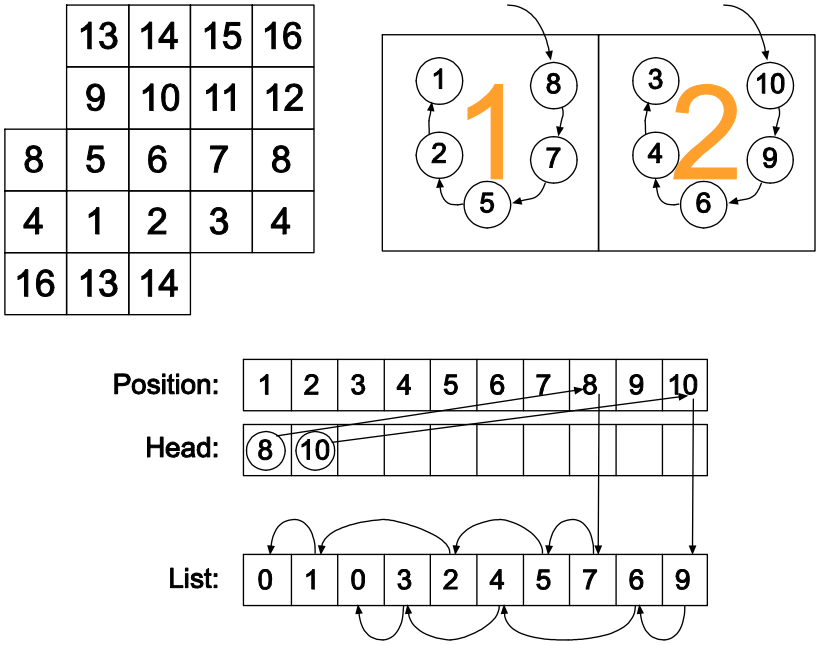

Figure 2.1.4: Linked-list method in two-dimensions, showing the atoms and the linked list structure.

Controlling the Simulation

In a basic molecular dynamics simulation, time evolution of N particles in a constant volume V is studied. In such simulations energy is conserved. This leads to micro-canonical (constant NVE) ensemble in statistical mechanics. It is desired to carry out simulations in different ensembles such as constant NVT (constant Number of particles, Volume and Temperature) and constant NPT (constant Number of particles, Pressure and Temperature). The temperature and pressure control is desired because it extends the applicability of MD simulations.

In a standard simulation, the density is controlled by the choice of the box volume V. Volume changes can be controlled by rescaling the volume upon request at the beginning of the simulation, then the pressure and temperature is measured during the calculation. There are advanced methods that the temperature and pressure can be controlled [23].

Molecular Dynamics at Constant Temperature If it is desired to investigate the system along a constant temperature rather than along a constant energy, the equations of motion have to be modified. In such a modification the system is cou-

pled to a heat bath, which introduces energy fluctuations necessary to keep a fixed temperature. Equilibrium of a system in a heat bath is a representative of canonical ensemble, where the number of particles N, the volume V, and the temperature T, are fixed and there is zero total linear momentum [33]. Although the total energy is not conserved in constant temperature simulations, average kinetic energy is a constant of motion due to its coupling with the temperature. In such a modification since the total energy is not conserved, important data is not collected in this stage. The controlled temperature simulations are used only to bring the system from one state to the other. Data collection is carried out after the system has reached equilibrium. There are various methods which are discussed separately in the accompanying sections, to fix the temperature to a fixed value during simulation.

Velocity Scaling In velocity scaling which is introduced by Woodcock [34], temperature is adjusted to the desired value by rescaling the velocities with a factor β. The scaling factor β is given in Equation 2.1.14.

$$\beta = \left[\frac{(3N-4)k_b T_{set}}{\sum_i m v_i^2} \right]^{1/2} \tag{2.1.14}$$

In Equation 2.1.14, T_{set} is the desired temperature and degrees of freedom is used as $3N-4$, instead of $3N$. The system has $3N$ degrees of freedom, because of zero total momentum in all directions, three degrees of freedom is removed and constant kinetic energy removes one degree of freedom.

Algorithm 3 *NVT molecular dynamics algorithm with velocity scaling*

do $t = 0, totstep$	start of MD loop
$r(t+\Delta t) = r(t) + v(t)\Delta t + \frac{1}{2}a(t)\Delta t^2$	calculate new positions
$v(t+\frac{\Delta t}{2}) = v(t) + \frac{1}{2}a(t)\Delta t$	calculate velocities at half step
$a(t+\Delta t) = -\frac{1}{m}\nabla V(r(t+\Delta t))$	evaluate force using new positions
$v(t+\Delta t) = v(t+\frac{\Delta t}{2}) + \frac{1}{2}a(t+\Delta t)\Delta t$	update velocities
$K = \frac{1}{2}mv(t+\Delta t)^2$	calculate kinetic energy
$\beta = [(3N-4)k_b T_{set}/2K]^{1/2}$	calculate the scale factor
$v(t+\Delta t) = v(t+\Delta t)\beta$	scale velocities
end do	

After scaling step we have

$$\frac{1}{2}\sum_i mv_i^2 = \frac{1}{2}(3N-4)k_bT_{set} \tag{2.1.15}$$

The algorithm for NVT molecular dynamics with velocity scaling using velocity Verlet integrator can be written as in Algorithm 3.

Berendsen Thermostat Berendsen [35] introduced a weak coupling to an external bath, using the principle of least local perturbation consistent with the required global coupling. In this method the desired temperature, T_{set}, is achieved by correcting the deviations of the actual temperature, T from T_{set} by scaling velocities with a parameter λ. The difference with respect to the velocity scaling is that the method allows for fluctuations of the temperature, not fixing it to a constant value [5].

$$\lambda = \left[1 + \frac{\Delta t}{\tau_T}\left(\frac{T_{set}}{T} - 1\right)\right]^{1/2} \tag{2.1.16}$$

In Equation 2.1.16 the constant τ_T is so called coupling time constant which determines the time scale on which the desired temperature is reached. Berendsen thermostat conserves Maxwell distribution, but the method does not resemble a specific thermodynamic ensemble.

Implementation of this method as suggested by Berendsen [35] was given in Algorithm 4.

Algorithm 4 *Constant temperature molecular dynamics with Berendsen thermostat*

Given $x(t)$ and $v(t-\frac{\Delta t}{2})$	
do $t = 0, totstep$	start of MD loop
$a(t) = -\frac{1}{m}\nabla V(r(t))$	evaluate forces
$T(t-\frac{\Delta t}{2}) = \frac{2K(t-\frac{\Delta t}{2})}{(3N-4)k_b}$	calculate instantaneous temperature
$\lambda = \left[1 + \frac{\Delta t}{\tau_T}\left(\frac{T_{set}}{T} - 1\right)\right]^{1/2}$	calculate Berendsen scale factor
$v(t+\frac{\Delta t}{2}) = v(t-\frac{\Delta t}{2}) + a(t)\Delta t$	update velocities
$v(t+\frac{\Delta t}{2}) = \lambda v(t+\frac{\Delta t}{2})$	scale velocities
$x(t+\Delta t) = x(t) + v(t+\frac{\Delta t}{2})\Delta t$	update positions
end do	

Nose-Hoover Thermostat The Nose method [36], considers an extended system within an additional degree of freedom s, which acts as an external heat reservoir, with an arbitrary mass Q, interacting with the system through the velocities of the particles. Hoover [37, 38], simplified the equations derived by Nose [39] where the equations of motion for this method is simplified by introducing a thermodynamical coefficient $\xi = s.p_s/Q$, where p_s is the conjugate momentum of s. Then, the reservoir itself is subject of an equation of motion which is simply a function of the kinetic energy of the system and the desired temperature,

$$\ddot{\xi} = \frac{1}{Q}\left[\sum_{i=1}^{N} m_i v_i^2 - N_f k_b T_{set}\right] \tag{2.1.17}$$

where N_f is the number of degree of freedom. The Hamiltonian conserved in the simulation is

$$H = K + V(r_N) + \frac{\dot{\xi}^2 Q}{2} + N_f k_b T \xi \tag{2.1.18}$$

A reversible algorithm for this method is given by Windiks [40], see Algorithm 5.

Algorithm 5 *NVT molecular dynamics algorithm with Nose-Hoover thermostat*

Given $\xi(t)$ position, $\dot{\xi}(t)$ velocity of thermostat	
$\dot{\xi}(t+\frac{\Delta t}{2}) = \dot{\xi}(t)+\frac{\Delta t}{2}(2K - N_f k_b T)/Q$	update thermostat velocity
$\xi(t+\Delta t) = \xi(t) + \dot{\xi}(t+\frac{\Delta t}{2})\Delta t$	update thermostat position
do $i = 1, N$	
$v_i(t+\frac{\Delta t}{2}) = v_i(t) \exp\left[-\dot{\xi}(t+\frac{\Delta t}{2})\frac{\Delta t}{2}\right] + a_i(t)\frac{\Delta t}{2}$	predict velocities
$r_i(t+\Delta t) = r_i(t) + v_i(t+\frac{\Delta t}{2})\Delta t$	calculate positions
$a_i(t+\Delta t) = -\frac{1}{m_i}\nabla V(r_i(t\Delta t))$	evaluate forces
$v_i(t+\Delta t) = \left[v_i(t+\frac{\Delta t}{2}) + a_i(t+\Delta t)\frac{\Delta t}{2}\right]$ $\exp\left[-\dot{\xi}(t+\frac{\Delta t}{2})\frac{\Delta t}{2}\right]$	correct velocities
end do	
$\dot{\xi}(t+\Delta t) = \dot{\xi}(t+\frac{\Delta t}{2})+\frac{\Delta t}{2}(2K - N_f k_b T)/Q$	update thermostat velocity

Molecular Dynamics at Constant Temperature and Pressure Some of the computation of certain quantities requires constant temperature and pressure ensemble such as the specific heat C_P at constant pressure. In addition to temperature

control, which is discussed in the previous section, if it is desired also to fix the pressure to a constant value, then the volume must be allowed to fluctuate [33]. Two common methods for this ensemble is discovered by Andersen and Berendsen, which are discussed in the following sections.

Andersen Method A summary of this method was given by Allen and Tildesley [26]. Andersen [41] introduced a method for controlling the pressure in molecular dynamics which involves coupling the system to an external variable V, the volume of the simulation box. This coupling act as a piston on a real system. The piston has a mass Q with kinetic energy

$$K_V = \frac{1}{2} Q \dot{V}^2 \tag{2.1.19}$$

The potential energy associated with the additional variable is

$$V_V = PV \tag{2.1.20}$$

where P is the specified pressure. In this method r and V are not coupled. The coordinates r_i of particles are replaced with the scaled coordinates ρ_i.

$$\rho_i = r_i / V^{1/3} \tag{2.1.21}$$

The equations of motion of the system are

$$\ddot{\rho} = \frac{F}{mV^{1/3}} - \frac{2\rho\dot{V}}{3V} \tag{2.1.22}$$

$$\ddot{V} = \frac{(P - P_{set})}{Q} \tag{2.1.23}$$

where Q is the piston mass, P is the instantaneous pressure, which can be calculated from the virial equation that is to be explained later (see Equation 2.1.32) and P_{set} is the desired pressure. The algorithm of this method was given in Algorithm 6.

Haile and Graben [42] described a way of implementing this method, solving the equations of motion in terms of the scaled positions and momenta in a box of unit length using a Gear predictor-corrector integrator.

The parameter Q, the piston mass, can be adjusted. A low mass results in rapid

box size oscillations and large mass gives rise to slow exploration of volume-space.

Algorithm 6 *NPT molecular dynamics algorithm with Andersen method*

Given box length L	
$\rho(t+\Delta t) = \rho(t) + \Delta t\dot{\rho}(t) + \frac{\Delta t^2 a(t)}{2L(t)}$	calculate positions
$V((t+\Delta t) = V(t) + \Delta t\dot{V} + \frac{\Delta t^2}{2Q}(P(t) - P_{set})$	calculate volume
$\dot{\rho}(t+\frac{\Delta t}{2}) = \dot{\rho}(t) + \frac{\Delta t}{2L(t)}a(t) - \frac{\Delta t}{2}\dot{\rho}(t)\frac{\dot{V}(t)}{V(t)}$	predict velocities
$\dot{V}(t+\frac{\Delta t}{2}) = \dot{V}(t) + \frac{\Delta t}{2Q}(P(t) - P_{set})$	predict volume velocity
$a_i(t+\Delta t) = -\frac{1}{m_i}\nabla V(r_i(t+\Delta t))$	evaluate forces
compute $P(t+\Delta t)$, and $L(t+\Delta t) = V(t+\Delta t)^{1/3}$	pressure and box size
$\dot{V}(t+\Delta t) = \dot{V}(t+\frac{\Delta t}{2}) + \frac{\Delta t}{2Q}(P(t+\Delta t) - P_{set})$	correct volume velocity
$\dot{\rho}(t+\Delta t) = \dot{\rho}(t+\frac{\Delta t}{2}) + \frac{\Delta t}{2L(t+\Delta t)}a(t+\Delta t)$ $-\frac{\Delta t}{2}\dot{\rho}(t+\frac{\Delta t}{2})\frac{\dot{V}(t+\frac{\Delta t}{2})}{V(t+\Delta t)}$	correct velocities
adjust temperature by velocity scaling	scale velocities

Algorithm 7 *NPT molecular dynamics algorithm with Berendsen method*

Given $x(t)$ and $v(t-\frac{\Delta t}{2})$	
do $t = 0, totstep$	
$a(t) = -\frac{1}{m}\nabla V(r(t))$	evaluate forces
calculate $P(t)$ from eq. 2.1.32	calculate instantaneous pressure
$\mu = \left[1 + \frac{\Delta t}{\tau_P}\beta(P(t) - P_{set})\right]^{1/3}$	calculate pressure scaling factor
$T(t-\frac{\Delta t}{2}) = \frac{2K(t-\frac{\Delta t}{2})}{(3N-4)k_b}$	calculate instantaneous temperature
$\lambda = \left[1 + \frac{\Delta t}{\tau_T}\left(\frac{T_{set}}{T} - 1\right)\right]^{1/2}$	calculate temperature scaling factor
$v(t+\frac{\Delta t}{2}) = v(t-\frac{\Delta t}{2}) + a(t)\Delta t$	calculate velocities
$v(t+\frac{\Delta t}{2}) = \lambda v(t+\frac{\Delta t}{2})$	scale velocities
$x(t+\Delta t) = x(t) + v(t+\frac{\Delta t}{2})\Delta t$	calculate positions
$x(t+\Delta t) = \mu x(t+\Delta t)$	scale positions
$L(t+\Delta t) = \mu L(t+\Delta t)$	scale box size
$V(t+\Delta t) = \mu V(t+\Delta t)$	scale volume
end do	

Berendsen Method Berendsen [35] pressure control method is similar to the Berendsen temperature control method discussed in previous section and involves

scaling of coordinates to adjust the volume, so the pressure. The coordinates are scaled by a factor given as

$$\mu = \left[1 + \frac{\Delta t}{\tau_P}\beta(P(t) - P_{set})\right]^{1/3} \tag{2.1.24}$$

where τ_P is the pressure coupling time and β is the isothermal compressibility. Berendsen [35] suggested a time constant for pressure of $0.1ps$ or larger. Using smaller time constants leads to instability of the algorithm with increased pressure and volume fluctuations. The algorithm of this method is given by Berendsen [35] as in Algorithm 7.

2.1.3 Property Calculations in Molecular Dynamics

In molecular dynamics, macroscopic quantities are calculated from the atomic positions and velocities using statistical mechanics. Statistical mechanical averages of energetic or structural properties are obtained as averages over the time steps [43]. The physical properties are generally a function of the particle coordinates and velocities. Therefore, an instantaneous value of a property A at a time t can be defined as

$$A(t) = f(r(t_i), ..., r(t_N), v(t_1), ..., v(t_N)) \tag{2.1.25}$$

and its average can be obtained as

$$\langle A \rangle = \frac{1}{N_{Tot}} \sum_{i=1}^{N_{Tot}} A(t_i) \tag{2.1.26}$$

where i is the index of the time step which ranges from 1 to total step number N_{Tot}. The property calculation in a molecular dynamics run, can be done in two different ways [23].

$A(t)$ can be calculated at each time step during the simulation, the sum is updated at each step and at the end of the simulation, the average is obtained by dividing the sum by the number of steps, or positions and velocities can be written to a file and at the end of the simulation, a separate program is used to process the trajectory and calculate the desired properties.

The most common physical properties calculated from molecular dynamics sim-

ulations are discussed in the following sections.

Potential Energy

Pair potential energy depends only on the magnitude of the separation between atoms. The average potential energy V is obtained by averaging its instantaneous values, calculated during force evaluation step. In case of pair potentials the potential energy is calculated by

$$V(t) = \sum_i \sum_{j>i} \varphi(|r_i(t) - r_j(t)|) \tag{2.1.27}$$

The potential energy gives an idea of energy conservation in a molecular dynamics run.

Kinetic Energy

The instantaneous kinetic energy is

$$K(t) = \frac{1}{2} \sum_i m_i \left[v_i(t)^2\right] \tag{2.1.28}$$

Total Energy

The total energy is a conserved quantity in Newtonian dynamics.

$$E = K + V \tag{2.1.29}$$

In order to check the conservation of it with time, total energy is monitored at each time step. Kinetic and potential energy fluctuate during the run while their sum remains fixed. In practice there could be small fluctuations in the total energy. These fluctuations are usually caused by errors in the time integration, and can be reduced in magnitude by reducing the time step. Berendsen & Van Gusteren [44] contains an in-depth analysis of total energy fluctuations using various time integration algorithms.

Temperature

Temperature, T, is directly related to kinetic energy by the familiar equipartition principle with an average energy of $k_b T/2$ per degree of freedom [26].

$$T = \frac{2K}{N_f k_b} \tag{2.1.30}$$

Temperature is obtained from the average kinetic energy using Equation 2.1.30. For practical purposes temperature is calculated instantaneously, using instantaneous kinetic energy, $K(t)$.

Pressure

Pressure is calculated by the Clausius virial equation:

$$W = -\sum_{i}^{N} \sum_{j>i}^{N} r_{ij} \cdot \frac{d\varphi(r)}{dr} \tag{2.1.31}$$

The pressure is defined as

$$PV = Nk_b T + \frac{1}{D} \langle W \rangle \tag{2.1.32}$$

where D is the dimensionality of the system (2, 3).

Radial Distribution Function

Radial distribution function, $g(r)$, is a measure for characterization of the local structure of a fluid. This function gives the probability of finding a pair of atoms a distance r apart, relative to the probability expected for a completely random distribution at the same density [26].

$$g(r) = \frac{V}{N^2} \left\langle \sum_{i} \sum_{j \neq i} \delta(r - r_{ij}) \right\rangle \tag{2.1.33}$$

Diffusion Coefficient

Diffusion coefficient can be estimated by the relation between mean square displacement and diffusion constant showed by Einstein.

$$3D = \lim_{t\to\infty} \frac{\left\langle |r(t) - r(0)|^2 \right\rangle}{2t} \tag{2.1.34}$$

Diffusion coefficient can be calculated by plotting the mean square displacement as a function of time and then attempting to obtain the limiting behavior. When using this method for calculating the diffusion coefficient, the mean squared distances should not be limited by the edges of the periodic box. In other words, a set of positions that were not corrected using periodic boundary conditions should be stored [45].

Another approach for calculating diffusion coefficient is to use autocorrelation functions. A correlation function provides a numerical value that encapsulates the data and quantifies the strength of the correlation. A molecular dynamics simulation provides data values at specific times. This enables the value of some property at some instant to be correlated with the value of the same or another property at a later time. When the correlated values are the same then the function is called an autocorrelation function. For instance velocity autocorrelation coefficient indicates how closely the velocity at time t is correlated with the velocity at time zero [45]. The value of the velocity autocorrelation coefficient can expressed as

$$C_{vv}(t) = \frac{1}{N} \sum_{i=1}^{N} v_i(t) \cdot v_i(0) \tag{2.1.35}$$

The velocity autocorrelation coefficient is related to diffusion coefficient by the Green-Kubo formula:

$$\int_0^\infty (v(\tau) \cdot v(0)) d\tau = \lim_{t\to\infty} \frac{\left\langle |r(t) - r(0)|^2 \right\rangle}{2t} = 3D \tag{2.1.36}$$

The long time-tail of the autocorrelation function can be dealt with by fitting a function to the curve and attempting to integrate to infinity.

2.2 Non-Equilibrium Molecular Dynamics

In the pervious section, the molecular dynamics simulation of systems under equilibrium conditions was considered. To improve the efficiency of the calculation of the transport properties and to examine directly the response of a system to a large perturbation real or imaginary using linear or nonlinear response theory, non-equilibrium molecular dynamics (NEMD) simulations are carried out. This approach is firstly introduced by Hoover and Ashurst [8]. In equilibrium molecular dynamics, properties represent the response to the naturally occurring small fluctuations. The signal-to-noise ratio is not favorable at long times. In addition, the finite system size limits time that correlation functions can be calculated reliably in equilibrium molecular dynamics (EMD) [26].

In contrast to EMD, NEMD introduce larger fluctuations artificially and the signal-to-noise level of the measured response is higher. In addition to that the steady state response is measured in NEMD. The calculations of transport properties in NEMD are more realistic and give better results than EMD [26].

For the transport coefficient of interest L_{ij}, that is

$$J_i = L_{ij} X_j \tag{2.2.1}$$

where J is the flux of some conserved quantity and X is a gradient in the density of that conserved quantity, Green-Kubo relation will be

$$L_{ij} = \int_0^\infty \langle J_i(t) \cdot J_j(0) \rangle \, dt \tag{2.2.2}$$

Invent a fictitious field F_e and its coupling to the system so that dissipative flux J_j holds

$$H_0^{ad} = -J_j F_e \tag{2.2.3}$$

where H_0^{ad} is the added term to the Hamiltonian due to perturbation. The increase in the Hamiltonian necessitates a thermostat, since otherwise, the work done on the system would be transferred continuously into heat and no steady state could be achieved. Then after the application of the thermostat, couple F_e to the system and compute the steady state average $\langle J_i(t) \rangle$, as a function of the external field F_e.

Linear response theory then proves,

$$L_{ij} = \lim_{F_e \to 0} \lim_{t \to \infty} \frac{\langle J_i(t) \rangle}{F_e}$$

This added an advantage to NEMD that it can be used to calculate non-linear as well as linear transport coefficients [9]. The application of NEMD methods to calculate transport coefficients is reviewed in detail by Evans and Morriss [46, 9] and Allen and Tildesley [26].

The calculation of self diffusion coefficients with NEMD was illustrated by Erpenbeck and Wood [47] for a system that exists a relation between susceptibility of the color current to the magnitude of the perturbing color field with the velocity autocorrelation function; where the diffusion coefficient can be written by Green-Kubo relation as:

$$D = \int_0^\infty dt \, \langle v_i(t) \cdot v_i(0) \rangle \tag{2.2.4}$$

The color Hamiltonian of the system can be written as:

$$H = H_0 - \sum_{i=1}^{N} c_i x_i F(t), \; t > 0 \tag{2.2.5}$$

where H_0 is the unperturbed Hamiltonian, c_i are the color charges, and an even number of atoms, N is considered with

$$c_i = (-1)^i \tag{2.2.6}$$

The considered response is the color current density, J_x,

$$J_x = \frac{1}{V} \sum_{i=1}^{N} c_i \dot{x}_i \tag{2.2.7}$$

Along with the lines of the above treatment and in the canonical ensemble,

$$\langle J_x(t) J_x(0) \rangle_c = \frac{1}{V^2} \sum_{i,j}^{N} c_i c_j \, \langle v_{xi}(t) v_{xj}(0) \rangle_c \tag{2.2.8}$$

Upon further simplification,

$$\langle J_x(t)J_x(0)\rangle_c = \frac{1}{V^2}\sum_{i=1}^{N} c_i^2 \langle v_{xi}(t)v_{xi}(0)\rangle_c = \frac{N}{V^2}\langle v_x(t)v_x(0)\rangle_c \tag{2.2.9}$$

Combining Equation 2.2.9 with Green-Kubo relation for self diffusion gives,

$$D = \frac{k_b T}{\rho}\lim_{t\to\infty}\lim_{F\to 0}\frac{\langle J_x(t)\rangle}{F} \tag{2.2.10}$$

If the total linear momentum was set to zero for a system in consideration, then v_{xi} is not independent of v_{xj}. Thus there is an order N^{-1} correction to this equation and the self diffusion coefficient is obtained as

$$D = \frac{N-1}{N}\frac{k_b T}{\rho}\lim_{t\to\infty}\lim_{F\to 0}\frac{\langle J_x(t)\rangle}{F} \tag{2.2.11}$$

The advantage of this equation is that just with the simple average of color current density, diffusivity can be predicted. It is important to note that this diffusivity does not give any information about the diffusion behavior under the applied force, since from Equation 2.2.11, the diffusivity is obtained when $F \to 0$. Thus the diffusivity obtained from this equation is the same as in the case when there is no applied force. In addition, in binary alloys this treatment is not applicable.

Chapter 3

Potentials

The most important part of an atomistic simulation is the potential function which describes the atomic interactions and the characteristic of the system. There are various approaches to obtain the potential function. The potential function can be obtained empirically, semi-emprically or from ab initio methods. Empirical potential functions are based on a simple analytical expression which may or may not be justifiable from theory, and which contains one or more parameters adjusted to an experimental situation [25]. Semi-empirical functions also contains parameters fitted to an experimental value but analytical expressions are derived from quantum-mechanical arguments.

Model pair potentials are derived within the pseudopotential context which are fit to specific experimental characteristics of atoms or solids and are used to analyze other physical characteristics or properties [48]. The application of pseudopotential method to the interactions between atoms is covered in this chapter.

3.1 Pseudopotential theory

The determination of the properties of metals requires a knowledge of interaction of electrons and ions. However, for many properties of materials, inner shell or core electrons are not explicitly relevant to the properties of the materials, but the valence electrons. The application of free-electron theory can be used to specify this interaction, where the ion and electron interaction is considered to be very weak. In contradiction with the free-electron theory, the ion core of a metal atom contains strong force fields which strongly affects nearby electrons.

At the electrons nearby an ion core, the strong Coulomb attraction of the ion

core is opposed by the repulsion due to the operation of the Pauli principle for the electrons of the closed shells. The net effective interaction experienced by an electron, as a result of the cancellation of the two principal contributions is quite small, and this interaction is known as *pseudopotential* [25].

The physical basis of the theory is therefore the replacement of the strong potential in the ion core, by creating an effective potential in which the nucleus and the core electrons together serve to influence the outer electrons, thus the interaction of electrons and ions are expressed with a weaker potential, see Figure 3.1.1. In the application of this method, it is important not to change the energy levels of the valence electrons. Thus the calculation of a pseudopotential involve the determination of the electron energies and wave functions. This requires a series of complex calculations to determine the potential parameters without any use of experimental data. There are several ab initio schemes developed up to now that deals with the generation of pseudopotentials with different flavours, few to mention are; Hamann-Schlüter-Chiang (HSC) [49, 50] , Rabe-Rappe-Kaxiras-Joannopoulos (RRKJ) [51], Troulier-Martins (TM) [52], Vanderbilt (US) [53] and Fritz-Haber Institute (FHI) [54] pseudopotentials. An earlier approach, namely the model potential approach, however involves much simpler calculations where the Fourier components of the metal are chosen to give some experimental properties of that element. The model pseudopotentials are derived from a physical basis and the parameters in the analytical expressions are obtained by fitting the expression to an experimental value.

3.1.1 Model Potentials

One of the simplest form of local model potential was introduced by Ashcroft [55]. It is defined in real space by

$$W(r) = \begin{cases} 0, & r \leqslant R_M \\ -2Z/r, & r > R_M \end{cases} \tag{3.1.1}$$

where Z is the electronic charge outside core region and R_M is the core radius which is determined by fitting the calculated values to experiment for such factors as the electrical resistance of liquid metals, data on the Fermi surface, and the equilibrium volume [48]. Bare-ion pseudopotential is defined in Fourier space as,

$$\langle k+q \,|W|\, k\rangle = W_0(q) = \frac{1}{\Omega}\int d^3r e^{-iqr} W(r) \tag{3.1.2}$$

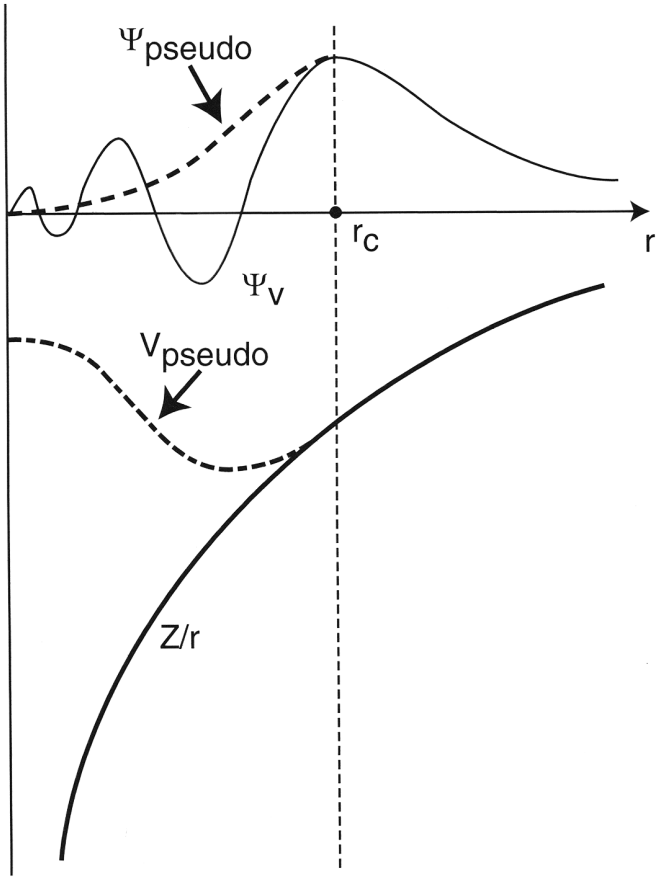

Figure 3.1.1: The pseudopotential formalism where real (solid lines) and pseudo (dashed lines) can be seen for the potential and the wave function.

where Ω is the atomic volume, r is the real space vector and q is the Fourier space vector. After Fourier transform, form factor is obtained as

$$W_0(q) = -\frac{4\pi Z}{\Omega q^2}\cos(qR_M) \tag{3.1.3}$$

The relationship between from factor and pseudopotential for Ashcroft potential can be seen in Figure 3.1.2. Core radii R_M for several elements were given in Table 3.1.1.

After Ashcoft several other schemes were developed to obtain such model potentials, few to mention are Heine-Abarenkov-Animalu [56, 57, 58], Harrison-Wills [59], Antonov-Milman [60], Fiolhais-Perdew [61], Bretonnet-Silbert [62, 63] and Hasegawa-Hoshino [64].

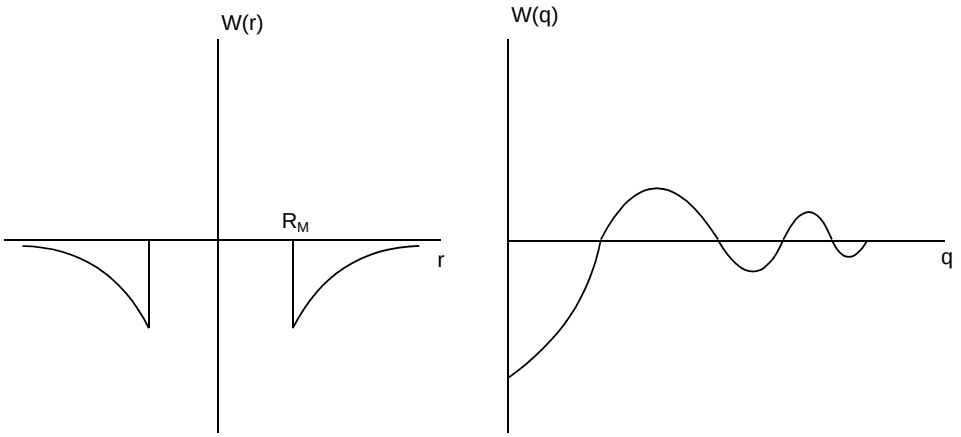

Figure 3.1.2: General form of the Ashcroft potential and its form factor

Table 3.1.1: Ashcroft empty core potential, ionic radii, units in Angstrom.

Element	Core Radii	Element	Core Radii	Element	Core Radii
Li	0.92	Zn	0.59	Sn	0.59
Be	0.58	Ga	0.59	Sb	0.56
Na	0.96	Ge	0.54	Te	0.54
Mg	0.74	As	0.51	Cs	1.55
Al	0.62	Se	0.5	Ba	1.6
Si	0.56	Rb	1.38	Hg	0.66
K	1.2	Cd	0.65	Tl	0.6
Ca	0.9	In	0.63	Pb	0.57
Bi	0.57	B	0.44	C	0.37
P	0.51	Sr	1.14	O	0.42
S	0.47	Cl	0.907		

Heine-Abarenkov pseudopotential

Heine-Abarenkov pseudopotentials are given separately for local and nonlocal parts [56, 57].

In real space the nonlocal pseudopotential considers each quantum number, l, see Figure 3.1.3, and has the form,

$$W(r) = \left\{ \begin{array}{ll} -\sum_{l=0}^{l_{\max}} A_l P_l & r \leqslant R_M \\ -2Z/r & r > R_M \end{array} \right\} \tag{3.1.4}$$

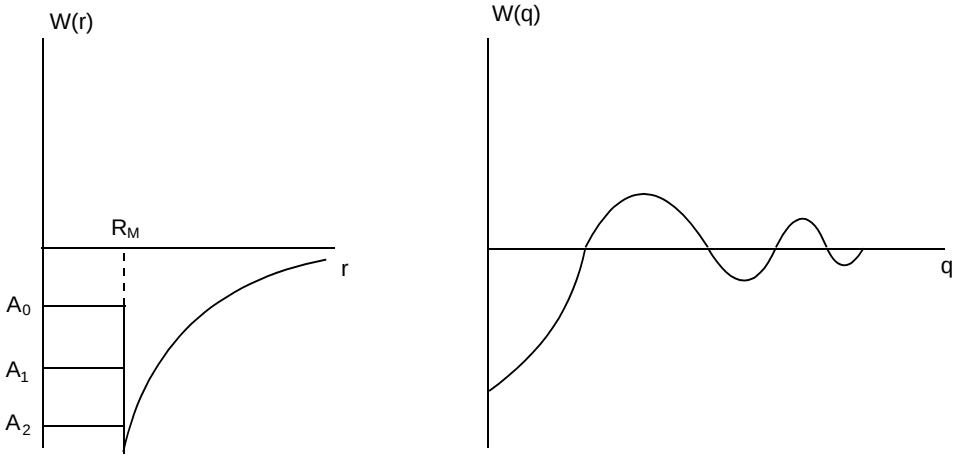

Figure 3.1.3: General form of the nonlocal Heine-Abarenkov pseudopotential and its form factor.

where the parameters A_l are obtained from spectroscopic data on the ionic energy levels and P_l are the projection operators.

The local pseudopotential takes the form

$$W(r) = \begin{cases} -A & r \leqslant R_M \\ -2Z/r & r > R_M \end{cases} \tag{3.1.5}$$

The bare ion pseudopotential in Fourier space with the formulation of Animalu [58] is given as

$$W_0(q) = N(q) + B(q) \tag{3.1.6}$$

where $N(q)$ and $B(q)$ are nonlocal and local parts and defined as

$$\begin{aligned} B(q) &= -\frac{8\pi C}{\Omega q^3}\left[\sin(qR_M) - qR_M\cos(qR_M)\right] - \frac{8\pi Z}{\Omega q^2}\cos(qR_M) \\ &\quad + \left(\frac{4\pi\left|E_c\right|}{\Omega q^3} - \frac{24\pi Z\alpha_{eff}}{\Omega q^2(qr_c)^3}\right) \times \left[\sin(qr_c) - qr_c\cos(qr_c)\right] \end{aligned} \tag{3.1.7}$$

For the nonlocal part, for $q \leqslant 2k_F$, where k_F is the Fermi wave vector and given as $k_F = \left(\frac{3\pi^2 Z}{\Omega}\right)^{1/3}$,

$$
\begin{aligned}
N(q) &= -\frac{4\pi}{\Omega}R_M^3(A_0-C)\left\{[j_0(x)]^2-\frac{1}{x}\cos(x)j_1(x)\right\} \\
&\quad -\frac{12\pi}{\Omega}R_M^3(A_1-C)\left\{[j_1(x)]^2-j_0(x)j_2(x)\right\}P_1(\cos\theta) \\
&\quad -\frac{20\pi}{\Omega}R_M^3(A_2-C)\left\{[j_2(x)]^2-j_1(x)j_3(x)\right\}P_2(\cos\theta)
\end{aligned} \tag{3.1.8}
$$

where $x = k_F R_M$, $\cos\theta = \left(1-\frac{q^2}{2k_F^2}\right)$ and θ is the angle between (k_F+q) and k_F wave vectors. P_n is the Legendre polynomial and given as

$$P_1(\cos\theta) = \cos\theta$$

$$P_2(\cos\theta) = \frac{1}{2}(3\cos^2\theta - 1)$$

$j_0(x), j_i(x), j_2(x), j_3(x)$ are zeroth, first, second and third order Bessel functions of the first kind respectively.

for $q > 2k_F$, the nonlocal part is given as

$$
\begin{aligned}
N(q) &= -\frac{8\pi}{\Omega(x^2-y^2)}R_m^3(A_0-C)\left\{xj_1(x)j_0(y)-yj_1(y)j_0(x)\right\} \\
&\quad -\frac{24\pi}{\Omega(x^2-y^2)}R_m^3(A_1-C)\left\{xj_2(x)j_1(y)-yj_2(y)j_1(x)\right\}P_1(\cos\theta') \\
&\quad -\frac{40\pi}{\Omega(x^2-y^2)}R_m^3(A_2-C)\left\{xj_3(x)j_2(y)-yj_3(y)j_2(x)\right\}P_2(\cos\theta')
\end{aligned} \tag{3.1.9}
$$

where $x = k_F R_M$, $y = (q-k_F)R_M$ and $\cos\theta' = \frac{(x^2+y^2-(qR_M)^2)}{2xy}$. All other parameters A_0, A_1, A_2, C, R_M, r_c, α_{eff} and E_c are adjustable parameters specific to the element in question.

Animalu [65, 66, 58] has parameterized all of them by fitting the potential to the experimentally obtained spectroscopic behavior of the element for many elements in the periodic table. The parameters for certain elements that were used in this work are given in Table 3.1.2. The form factor obtained for aluminum using Heine-Abarenkov-Animalu approach was given in Figure 3.1.4.

Table 3.1.2: Heine-Abarenko-Animalu potential parameters for the elements that were used in this study.

Element	A_0	A_1	A_2	C	R_M	r_c	α_{eff}	E_c
Na	0.305	0.339	0.402	0.402	3.4	1.85	0.052	0.072
Mg	0.78	0.88	0.99	0.99	2.6	1.47	0.043	0.086
Al	1.38	1.64	1.92	1.92	2.0	1.08	0.024	0.094
Ti	2.3	2.5	2.1	2.0	2.0	1.285	0.037	0.096
Mn	0.89	0.98	0.87	0.91	2.2	1.512	0.088	0.095
Cu	0.25	0.4	0.215	0.455	2.2	1.814	0.157	0.086
Zn	0.99	1.14	0.98	0.91	2.2	1.57	0.079	0.091
Sn	1.84	2.04	1.62	1.62	2.0	1.4	0.032	0.092

3.1.2 Screening of the Pseudopotential

The electric field created from the ion, polarizes the surrounding charges and this causes the redistribution of the electron density. This new electron density distribution, as a consequence, causes a weaker electric field to be created than the electric field created from a bare ion. This is called electron screening. The screening effect on the bare ion pseudopotentials can be treated as dividing the potentials' Fourier constants to the modified Hartree dielectric function.

If the electron-electron interaction is ignored, ideal Fermi electron gas condition was obtained and the use of the quantum mechanical perturbation theory becomes possible that yields the electron response function, which can be written as

$$\begin{aligned} \chi_0(q) &= -\frac{mk_F}{\hbar^2\pi^2}\left(\frac{1-x^2}{4x}\ln\left|\frac{1+x}{1-x}\right| + \frac{1}{2}\right) \\ x &= q/2k_F \end{aligned} \tag{3.1.10}$$

This function is the Lindhard [67] free electron polarization function. Thus, the free electrons under the effective potential δV_{eff} have a disturbed electron density given by

$$\delta n(q) = \chi_0(q)\delta V_{eff}(q) \tag{3.1.11}$$

As it can be seen from the Equation 3.1.11, there is a linear response. In the case of an external field, the effective potential on the electrons is given by

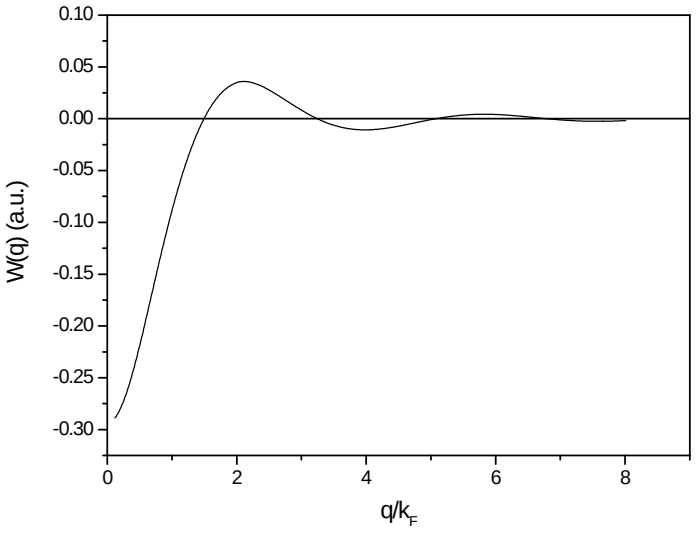

Figure 3.1.4: Heine-Abarenkov pseudopotential form factor for Al.

$$\begin{aligned} \delta V_{eff}(q) &= \delta V_{ext}(q) + V_c(q)[1 - G(q)]\delta n(q) \\ V_c(q) &= \frac{4\pi e^2}{q^2} \end{aligned} \tag{3.1.12}$$

where $V_c(q)$ is the bare Coulomb interaction between electrons and $G(q)$ is the local field correction factor (LCF) which involves the effects originated from the electron-electron correlation on the Coulomb interaction. If Equations 3.1.11 and 3.1.12 are solved simultaneously, the below Equation is obtained.

$$\delta n(q) = \frac{\chi_0(q)\delta V_{ext}(q)}{1 - V_c(q)[1 - G(q)]\chi_0(q)} \tag{3.1.13}$$

From this point, Hartree dielectric function can be obtained as

$$\begin{aligned} \delta n(q) &= \frac{\chi_0(q)\delta V_{ext}(q)}{1 - V_c(q)[1 - G(q)]\chi_0(q)} \equiv -\frac{1}{V_c(q)}\left(1 - \frac{1}{\varepsilon(q)}\right)\delta V_{ext}(q) \\ \varepsilon(q) &= 1 - \frac{V_c(q)\chi_0(q)}{1 + V_c(q)\chi_0(q)G(q)} \end{aligned} \tag{3.1.14}$$

As a consequence the screened pseudopotentials can be found in the following way

$$\begin{aligned} W(q) &= \frac{W_0(q)}{\varepsilon^*(q)} \\ \varepsilon^*(q) &= 1 - \frac{1}{\varepsilon(q)} \end{aligned} \tag{3.1.15}$$

The effect of LCF is very profound therefore, there were numerous studies in the literature. The local field correction factors used in this study are given in the following sections.

RPA (random phase approximation)

$$G(q) = 0 \tag{3.1.16}$$

Hubbard [68]

$$G(q) = \frac{q^2}{2(q^2 + k_F^2)} \tag{3.1.17}$$

Kleinman [69]

$$\begin{aligned} G(q) &= \frac{1}{4}\left(\frac{q^2}{q^2 + \xi k_F^2} + \frac{q^2}{\xi k_F^2}\right) \\ \xi &= \frac{2}{1 + 0.02544716775 r_s} \end{aligned} \tag{3.1.18}$$

where r_s is the pair interaction factor between electrons and it is related to electron density, $n = \Omega/Z$, with

$$r_s = \frac{me^2}{\hbar^2}\left(\frac{3}{4}\pi n\right)^{1/3} \tag{3.1.19}$$

Geldart-Vosko [70]

$$G(q) = \frac{1}{2}\left(\frac{q^2}{q^2 + \xi k_F^2}\right) \tag{3.1.20}$$

Langreth [71]

$$
\begin{aligned}
G(q) &= \frac{1}{4}\left(\frac{q^2}{q^2+k_F^2+K_s^2}+\frac{q^2}{k_F^2+K_s^2}\right) && (3.1.21)\\
K_s^2 &= \frac{k_F^2(1-\varsigma)}{\varsigma}\\
\varsigma &= \frac{1}{2}\left[1+0.158\left(\frac{k_{TF}}{2k_F}\right)^2\right]\\
k_{TF} &= \frac{2k_F}{\pi} && (3.1.22)
\end{aligned}
$$

Ichimaru-Utsumi [72]

$$
\begin{aligned}
G(Q) &= AQ^4+BQ^2+C+\left[AQ^4+(B+\frac{8}{3}A)Q^2-C\right] && (3.1.23)\\
&\quad \times\left(\frac{4-Q^2}{4Q}\right)\ln\left|\frac{2+Q}{2-Q}\right|\\
Q &= \frac{q}{k_F}\\
A &= 0.029\\
B &= \frac{9}{16}\gamma_0-\frac{3}{64}\left[1-g(0)\right]-\frac{16}{15}A\\
C &= -\frac{3}{4}\gamma_0-\frac{9}{16}\left[1-g(0)\right]-\frac{16}{5}A
\end{aligned}
$$

where $g(0)=\frac{1}{8}\left(\frac{z}{I_1(z)}\right)^2$, $z=4\sqrt{\frac{\alpha r_s}{\pi}}$, $\alpha=\left(\frac{4}{9}\pi\right)^{1/3}$

$I_1(z)$ is first order modified Bessel function, γ_0, is related to electron correlation energy and compressibility.

$$
\begin{aligned}
\gamma_0 &= \frac{1}{4}-\frac{\pi\alpha}{24}r_s^5\frac{d}{dr_s}\left[\frac{1}{r_s^2}\frac{d}{dr_s}E_c(r_s)\right]\\
\frac{d}{dr_s}E_c(r_s) &= \frac{b_0}{r_s}\left[\frac{1+b_1x}{1+b_1x+b_2x^2+b_3x^3}\right],\ x\equiv\sqrt{r_s}
\end{aligned}
$$

where $b_0=0.0310907$, $b_1=9.81379$, $b_2=2.82224$, $b_3=0.736411$

Vashishta-Singwi [73]

$$G(q)=A[1-\exp(-BQ^2)], Q=\frac{q}{k_F} \qquad (3.1.24)$$

r_s	1	2	3	4	5	6
A	0.70853	0.85509	0.97805	1.08482	1.17987	1.26569
B	0.36940	0.33117	0.30440	0.28430	0.26850	0.25561

For the intermediate electron liquids, the below fit equations are used,

$$A(x) = a + bx + cx^3 + d\sqrt{x} + \frac{e}{x} \tag{3.1.25}$$

where $a = 0.287730339$, $b = 0.012549482$, $c = -4.929 \times 10^{-5}$, $d = 0.370269366$, $e = 0.038030103$

$$B(x) = a + bx + cx^3 + d\ln x + e\frac{\ln x}{x^2} \tag{3.1.26}$$

where $a = 0.372944449$, $b = -0.00355839$, $c = 1.39398 \times 10^{-5}$, $d = -0.05599457$, $e = 0.026795153$

Moronoi-Ceperley-Senatore [74]

$$G(q) = \left\{ \left[(A-C)^{-n} + \left(\frac{Q^2}{B}\right)^n \right]^{-1/n} + C \right\} Q^2,\ Q = \frac{q}{k_F} \tag{3.1.27}$$

where $n = 8.014155308 - 0.00018224\exp(r_s)$

$A = \gamma_0$ (same as Ichimaru-Utsumi)

$B = \frac{1+2.15x+0.435x^3}{3+1.57x+0.409x^3}$

$C = \frac{\pi}{2e^2k_F}\left[-\frac{d}{dr}(r_sE_c(r_s))\right]$

$x = \sqrt{r_s}$

The forms of local field corrections for aluminum that are covered in this chapter were given in Figure 3.1.5.

3.2 Pair Potentials Derived by Pseudopotential Theory

The total energy of the system per ion involves two parts. The first part is the volume term and consists the largest part, however it is independent from the position of the ions. The second part is composed of free electron gas energy, electrostatic energy

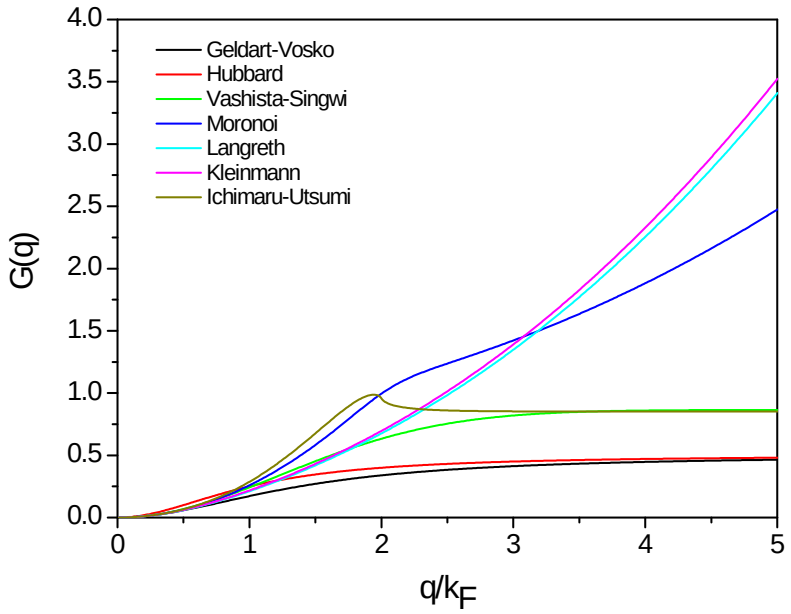

Figure 3.1.5: Screening functions for Aluminum.

and band energy due to ion-ion, electron-ion and electron-electron interaction. The second order perturbation theory defines the band structure energy as

$$U_{bs} = \sum_{g} |S(g)|^2 \, \Phi_{bs}(g) \tag{3.2.1}$$

where g is the reciprocal space lattice vector, $S(g)$ is the structure factor which determines the ion distribution and $\Phi_{bs}(g)$ is called as energy wave number characteristic or characteristic function of band structure energy. The Equation 3.2.1 can be rewritten as

$$U_{bs} = \frac{1}{N^2} \sum_{\nu\mu} \Phi_{bs}(q) e^{iq(t_\nu - t_\mu)} \tag{3.2.2}$$

In Equation 3.2.2 a sum is done over reciprocal space lattice positions and ion types. If $\varphi_{bs}(r)$ is defined as the Fourier transform of $\Phi_{bs}(q)$,

$$\varphi_{bs}(r) = \frac{2\Omega}{(2\pi)^3} \int d^3q \Phi_{bs}(q) e^{iqr} \tag{3.2.3}$$

then the band structure energy can be rewritten as the sum of the pair interaction potentials between atoms in real space.

$$U_{bs}(r) = \frac{1}{2N} \sum_{\nu\mu} \varphi_{bs}(r_\nu - r_\mu) \tag{3.2.4}$$

$\varphi_{bs}(r)$ determines the pair ion interaction through the conducting electrons, therefore it is an indirect interaction. If the direct Coulomb interaction is added to this term, then the effective pair potential between the ions is obtained.

$$\varphi(r) = \frac{Z^2 e^2}{r} + \frac{\Omega}{\pi^2} \int_0^\infty dq \Phi_{bs}(q) \frac{\sin(qr)}{qr} \tag{3.2.5}$$

The energy wave number characteristic is independent of the ion distribution and depends only to the pseudopotential form factor and the atomic volume. In addition, in the calculation of the characteristic function of the band structure energy, screening effect of the pseudopotential form factors must be considered. Therefore the effective pair interaction potential between ions explicitly can be given as

$$\begin{aligned} \varphi(r) &= \frac{Z^2 e^2}{r} \left\{ 1 - \frac{2}{\pi} \int_0^\infty dq \left[1 - \frac{1}{\varepsilon} \right] M^2(q) \frac{\sin(qr)}{qr} \right\} \\ M^2(q) &= \left[\frac{\Omega q^2}{4\pi e^2} \right]^2 |W_0(q)|^2 \end{aligned} \tag{3.2.6}$$

The interaction potential using Equation 3.2.6 for aluminum using Ashcroft and Heine-Abarenkov-Animalu pseudopotentials with Geldart-Vosko screening function were given in Figure 3.2.1.

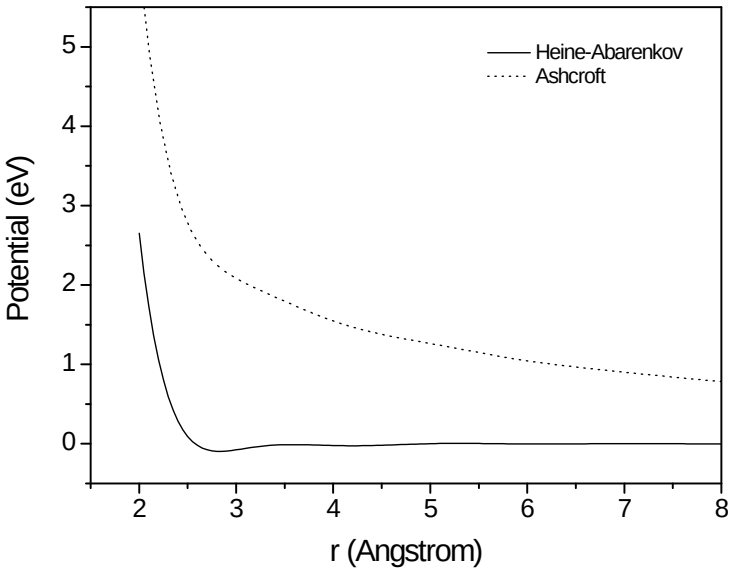

Figure 3.2.1: Pair interaction potentials derived for Aluminum using Ashcroft and Heine-Abarenkov-Animalu pseudopotentials with Geldart-Vosko screening function.

Chapter 4

Pseudopotential based Electromigration theory

4.1 Introduction

The electromigration theory is reviewed in aspects of thermodynamics and kinetics by Ho and Kwok [75]. Electromigration is a basic diffusion process under a driving force. There is phenomenological and theoretical approaches for the determination of this driving force in the literature [76]. It is found that the electromigration driving force is related to the direction of motion of the electrons which is then called electron wind force.

Electromigration driving force consists of two parts, one of which is the force from the applied electric field and the other is the electron scattering force. Fiks [77] and Huntington and Grone [78] formulated the electron wind driving force using a ballistic approach to handle the collision of electrons with the atoms. Huntington and Grone found that the driving force depends on the atomic configuration of the diffusion path and the types of the defects. However there is a problem of seperating the atoms, lattice and electrons in this method [14]. A quantum-mechanical model was established by Bosvieux and Friedel [79], in which the electron scattering force is taken as electrostatic force of the electron charge acting on the scattering center using weak electron-ion Coulumb potentials.

Sorbello [14, 15, 16] has made a very comprehensive study about determining the driving force for electromigration for each individual atom in a lattice with a pseudopotential method. Sorbello reported that the electromigration force on each

atom consists of two different forces, one of which is the force from the applied electric field and the other is the force exerted by the scattering electrons. It was reported that the electron scattering force is caused by the impurities and vacancies in the lattice.

In recent studies, Todorov et al [80] calculated the forces on each atom due to an applied electric field by a self-consistent tight binding formalism. Dekker and Lodder [81] has calculated the electromigration wind force in FCC and BCC metals with a Green's function approach.

In this study an atomistic approach to electromigration induced diffusion in metals is aimed to be simulated with the use of the pseudopotential method driven by Sorbello [14], therefore a detailed discussion of this method is given in the following section.

4.2 Pseudopotential Based Electromigration Driving Force

Sorbello [14] used a similar method with Bosvieux and Friedel [79], that is the electron scattering force is calculated by the classical electrostatic force of the electron charge acting on the ion. However for the electron-ion interaction, instead of weak Coulomb potential, Sorbello suggested a pseudopotential method for the calculation of the forces and charge densities.

Sorbello handled the force calculation in two separate parts; force on an ion arising from the electric field and from the electric current which is the electron scattering force.

Sorbello used the Born-Oppenheimer approximation to express the electron dependent force on an ion j:

$$F_j = \int n(r) \frac{\partial V_0}{\partial R_j} dr \tag{4.2.1}$$

where $n(r)$ is the electron density, V_0 is the bare electron-ion interaction potential and R_j is the ion position. Then Sorbello obtained the screened current dependent charge density, and instead of electron-ion potential used a pseudopotential and modified Equation 4.2.1 after some derivation [14] as

$$F_j = -TR\rho \frac{\partial W_0}{\partial R_j} \tag{4.2.2}$$

where ρ is given as the current dependent part of the density operator and W_0 is the unscreened pseudopotential. After some algebra, Sorbello expressed the electron scattering force on an ion in terms of a potential U_j:

$$F_j = -\frac{\partial U_j}{\partial R_j} \tag{4.2.3}$$

where U_j is given by

$$U_j = -\sum_{\substack{n \\ vacancies}} u_{jn} + \sum_{\substack{i \\ impurities}} u_{ji} \tag{4.2.4}$$

and u_{jm} is defined as

$$u_{jm} = \left\{ \begin{array}{c} -\frac{\Omega^2 m k_F}{4\pi^3\hbar^2}\left(\frac{v_d}{v_F}\right)\cos(v_d, R_j - R_m)\int_0^{2k_F} W_j(q)W_m(q)q^2 j_1(q\,|R_j - R_m|)dq \\ \text{for } R_j \neq R_m \\ -\frac{\Omega^2 m k_F}{12\pi^3\hbar^2}\left(\frac{v_d \cdot R_j}{v_F}\right)\int_0^{2k_F} W_j(q)W_m(q)q^3 dq, \text{ for } R_j = R_m \end{array} \right\} \tag{4.2.5}$$

where v_d is the electron drift velocity due to applied electric field, v_F is the Fermi velocity, $W_j(q)$ is the form factor of specie j, Ω is atomic volume, j_1 is the spherical Bessel function defined as $j_1(x) = x^{-2}\sin x - x^{-1}\cos x$.

If the above expression is applied to a perfect lattice of atoms, the contribution to the driving force is obtained as zero. Sorbello has investigated the above expression for different diffusion mechanisms such as vacancy or interstitial mechanisms.

In the absence of distortion the force on a substitutional impurity, obtained by regarding the substitutional impurity consist of an impurity and a lattice vacancy occupying the same site, is given by

$$F_{SUB} = \frac{\Omega^2 m k_F}{12\pi^3\hbar^2}\left(\frac{v_d}{v_F}\right)\int_0^{2k_F} W_2(q)\left[W_2(q) - W_1(q)\right]q^3 dq \tag{4.2.6}$$

where $W_2(q)$, and $W_1(q)$ is the form factor of substitutional impurity and host ion respectively. In addition, the force on an atom that derives it to jump to a nearest neighbor vacancy was given as

$$F_{VAC} = -\frac{\Omega^2 m k_F}{4\pi^2\hbar^2}\left(\frac{v_d}{v_F}\right)\left\{\frac{2}{\lambda}\int_0^{2k_F} W_1(q)^2 q^2 j_1(q\lambda)dq - \frac{1}{3}\int_0^{2k_F} W_1(q)^2 q^3 dq\right\} \tag{4.2.7}$$

where λ is the nearest neighbor jump distance.

Chapter 5

Methodology

In this study, bulk diffusion characteristics of aluminum alloys under electromigration force was investigated using molecular dynamics method. For this purpose a molecular dynamics program was developed. The program was written in FORTRAN language. The structure of the developed program is given in Figure 5.0.1.

At the initialization part as mentioned earlier, initial atomic positions and velocities were assigned to atoms and the simulation parameters such as temperature and number of MD steps were input to the program. 8000 Al atoms were put into a box of $40.5 \times 81 \times 40.5$ Angstrom MD box where there are actually $10 \times 20 \times 10$ unit cells of FCC lattice along the x, y and z Cartesian coordinates as schematically shown in Figure 5.0.2. In order to observe the diffusion behavior under electromigration force, an electric field is applied along the positive x-axis direction. The application of the electric field in x direction, as a consequence of electromigration force, causes the movement of the entire cell in that direction. This bulk mass transfer is similar to convection phenomena. Actually, there is no convection in interconnects, because of the boundary conditions imposed on the real geometry. Hence, to avoid this, one cell from the top and one cell from the bottom along the y-axis were fixed. The fixed atoms are designated with red color in Figure 5.0.2. This results 800 atoms to be fixed, 400 from the top and 400 from the bottom. The introduction of fixed atoms in the simulation cell hinder the transport of atoms that are nearby the fixed atoms. In order to overcome this falsifying effect on the diffusion behavior a rather larger simulation cell was constructed along the y direction and only the atoms placed in the middle of the box (which makes 4000 atoms) were analyzed for the determination of the diffusion properties. The system of atoms that were analyzed was shown in Figure 5.0.2. In addition, 32 atoms were taken out to obtain vacancies in the system.

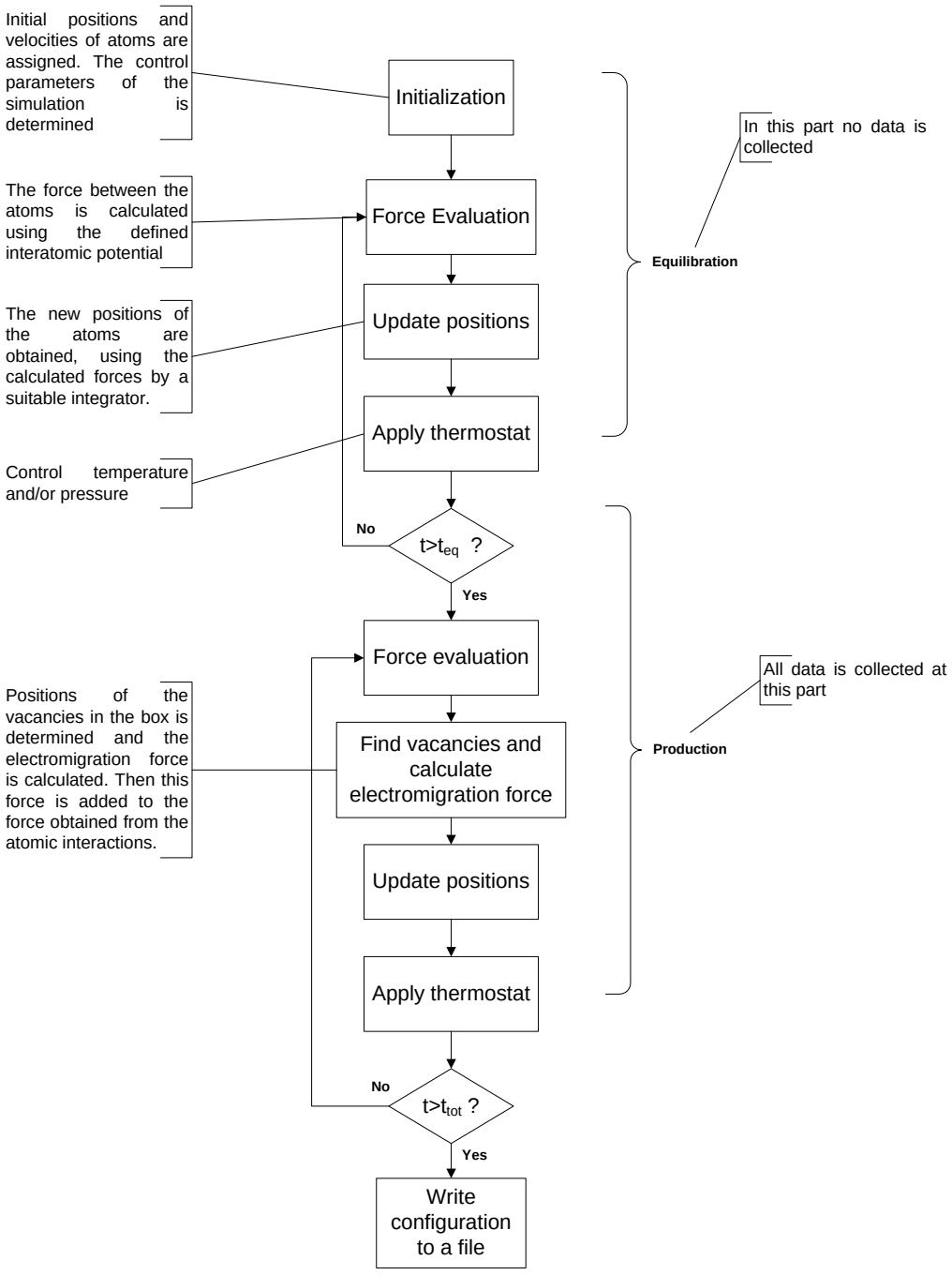

Figure 5.0.1: Molecular dynamics program structure.

These 32 vacancies were placed in a mesh, at the center part of the cell. As can be seen from the Figure 5.0.3, the vacancies were located as groups of 4 vacancies. This is introduced to the system in order to increase the electromigration force on nearby atoms so that diffusion becomes appreciable. 45 substitutional atoms which are Cu, Mg, Mn, Sn, Ti, are also placed at the center part of the cell in a mesh. In Figures 5.0.2 and 5.0.3 the distribution of substitutional atoms in the starting cell is shown.

Initial velocities were given to the atoms randomly from a Gaussian distribution. At this stage the total momentum of the system was set to be zero. In addition, the velocities were scaled to adjust the temperature to the desired value. All of the control parameters of the MD simulation were given to the program with an input file.

For the integration of the Newton's equation of motion, Gear predictor-corrector and velocity Verlet algorithms were implemented in the program. In this work velocity Verlet algorithm was used, since Verlet algorithm is more attractive in terms of energy conservation at longer time steps. In addition, since there is a large number of atoms used in the simulation box, in order to reduce the computation time, linked list algorithm was implemented in the program.

Heine-Abarenkov-Animalu [56, 57, 58] model potentials were employed for the atomic interactions. The scheme used for interatomic potential generation, as was finalized in Equation 3.2.6 gives the results at discrete points in real space. In order to avoid lengthy computations every time in MD force evaluations, the potential should have an analytical form. Therefore data was fitted to a 13 parameter Fourier series given in Equation 5.0.1 and used in this analytical form in the molecular dynamics program.

$$\begin{aligned}\varphi(r) &= a + b\cos(\pi\frac{r-r_0}{r_n-r_0}) + c\sin(\pi\frac{r-r_0}{r_n-r_0}) \\ &+ d\cos(2\pi\frac{r-r_0}{r_n-r_0}) + e\sin(2\pi\frac{r-r_0}{r_n-r_0}) \\ &+ f\cos(3\pi\frac{r-r_0}{r_n-r_0}) + g\sin(3\pi\frac{r-r_0}{r_n-r_0}) \\ &+ h\cos(4\pi\frac{r-r_0}{r_n-r_0}) + i\sin(4\pi\frac{r-r_0}{r_n-r_0}) \\ &+ j\cos(5\pi\frac{r-r_0}{r_n-r_0}) + k\sin(5\pi\frac{r-r_0}{r_n-r_0}) \\ &+ l\cos(6\pi\frac{r-r_0}{r_n-r_0}) + m\sin(6\pi\frac{r-r_0}{r_n-r_0})\end{aligned} \tag{5.0.1}$$

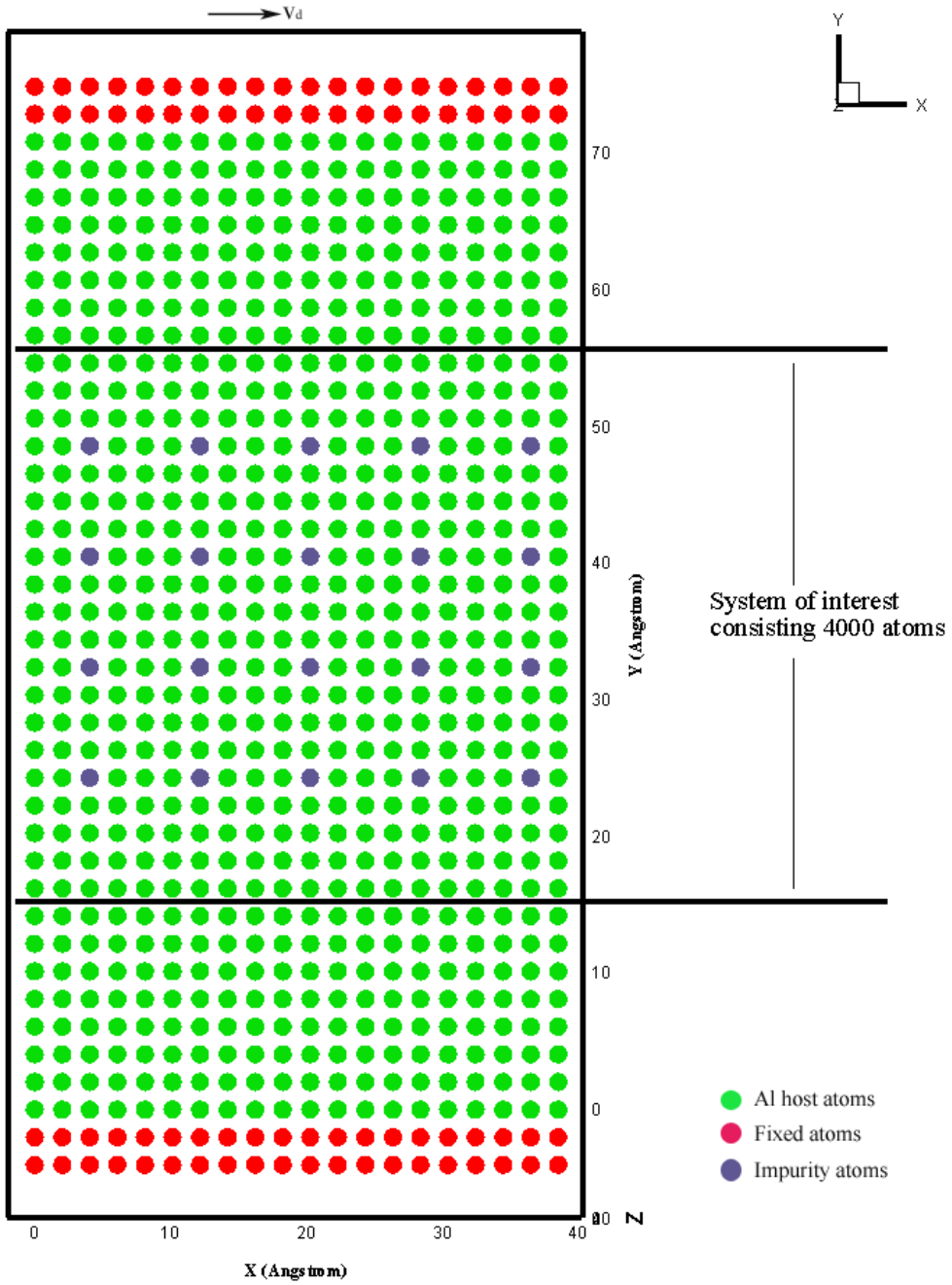

Figure 5.0.2: Schematic diagram of MD cell in two dimensions, used in this work, representing the distribution of host, fixed and impurity atoms.

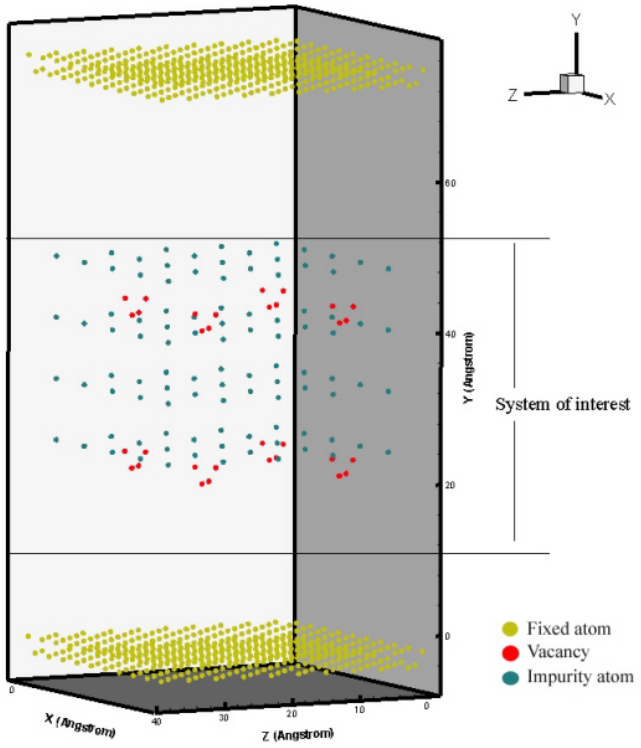

Figure 5.0.3: Atomic configuration of the impurities, vacancies and fixed atoms when the host atoms are excluded.

where a to m are the Fourier coefficients and r_0 and r_m gives the range for real space vector where fitting was carried out. The fitting of the Al-Al pair potential data to Fourier series gives a goodness of fit of $r^2 = 0.9999933576$ which is exceptionally good and represents the data with at most 1 meV error in the potential. This value is within the acceptable tolerance in ab initio calculations. The use of more complex functions increases the computation time unnecessarily. The Fourier fit parameters of the potentials of elements used in this study were given in Table 5.0.1, as well as the potential itself in Figure 5.0.4.

The evaluation procedure of potential functions for different screening functions were carried out using GULP [82] software. The results of the interatomic potential evaluation was given in the forthcoming chapter.

For the temperature control of the simulations, velocity scaling, Berendsen method and Nose-Hoover thermostat [36, 37], and for the pressure control Andersen [41] and Berendsen [35] methods were implemented in the program. Molecular dynamics simulations accomplished in this study were carried out in NVT ensemble. As men-

Table 5.0.1: Fourier fit parameters for interatomic potential for all pair interactions. The range of the potentials was carried out between 2.0 and 8.0 Angstrom, respectively as r_0 and r_n. The units of all of the Fourier parameters are in eV.

	Al - Al	Al - Cu	Al - Mg	Al - Mn	Al - Sn	Al - Ti
a	28.58820	35.443622	11.803691	20.846359	44.449366	48.420602
b	11.37887	11.421738	6.9773471	8.418488	16.066938	14.079163
c	-49.43655	-61.82470	-19.54810	-35.95302	-77.14364	-85.17685
d	-31.58869	-40.82441	-10.66685	-22.84379	-50.03738	-57.50072
e	-15.35909	-15.31378	-8.982112	-11.24089	-21.56163	-19.48912
f	-11.56402	-11.55592	-6.113529	-8.369197	-16.21126	-15.38738
g	14.41640	20.106552	3.216899	10.402513	23.732942	29.093363
h	4.41000	7.159457	0.099842	3.207555	7.871105	10.519685
i	5.44469	5.600885	2.345433	3.928827	7.740661	7.825999
j	1.50859	1.688394	0.417812	1.099452	2.229726	2.434664
k	-0.81759	-1.677667	0.238918	-0.599425	-1.669391	-2.488076
l	-0.08478	-0.215718	0.042433	-0.060328	-0.194297	-0.295892
m	-0.196421	-0.244103	-0.009577	-0.142551	-0.303402	-0.368709

tioned earlier, NPT simulations are more realistic, however since in the construction of simulation box experimental lattice parameter for aluminum was used, a realistic pressure was obtained in the NVT ensemble with some fluctuation. Thus in order to save computer time, simulations were carried out in NVT ensemble. Temperature control was accomplished by Nose-Hoover thermostat and the thermostat mass of $Q = 50$ amu was used in this work. The simulations were carried out at temperatures of 1000 K, 1200 K and 1400 K. The melting temperature was calculated for aluminum with the generated potential as 1800 K. In literature Tu [83] was pointed out that the electromigration failure occurred in aluminum by lattice diffusion was seen at $\frac{3}{4}$of the melting temperature.

In the equilibration part of the simulation electromigration force was not applied to the atoms. At this stage the system is left to reach to thermal equilibrium. In this study 20000 steps of equilibration was used since in non-equilibrium molecular dynamics method the equilibration stage is not critical. Total molecular dynamics run proceeded 60000 steps further with $\Delta t = 2$ fs. One important fact should be kept in mind that, in order for the time unit to be seconds, SI units should be used all throughout the simulation where mass should be in kg and distance should be in meters, then energy will be in Joules. However the use of SI units in atomistic simulations necessitates to deal with very small numbers which might cause sum errors in the calculations due to insignificant number of digits. Therefore it is wise to use atomic units or preferably the amu (mass), Angstrom (distance), eV (energy)

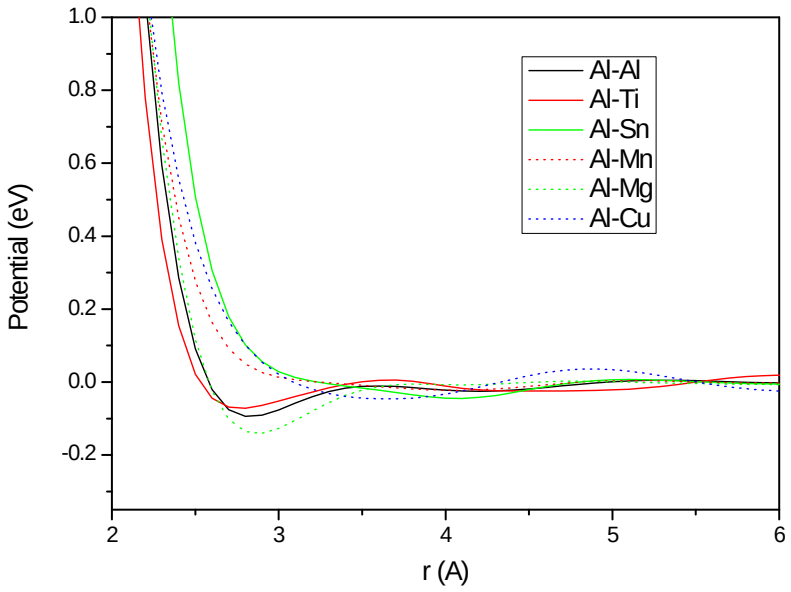

Figure 5.0.4: Pair potentials between aluminum and alloying elements used in this study.

unit system. In this case what will be the unit of time ? It can be evaluated that if the arbitrary unit of time is multiplied by a factor of 0.09822305118, the unit of time will be femtoseconds.

In the calculation of the electromigration wind force, the drift velocity of electrons due to the applied electric field has to be specified. In the microscopic view of the Ohm's Law it is suggested that the proportionality of current and voltage comes from the fact that an applied electric field superimposes a small drift velocity on the free electrons in a metal. For ordinary currents, this drift velocity is on the order of millimeters per second in contrast to the speeds of the electrons themselves which are on the order of a million meters per second.

The drift velocity can be expressed in terms of the accelerating electric field E, the electron mass m, and the characteristic time between collisions τ as,

$$v_d = \frac{eE}{m}\tau = \frac{eE}{m}\frac{d}{v_F} \tag{5.0.2}$$

where e is the electronic charge, d is the mean free path of an electron and v_F is the Fermi velocity. In addition, the conductivity of the metal can be expressed in terms

of the density of the free electrons, n, as

$$\sigma = \frac{ne^2 d}{mv_F} \tag{5.0.3}$$

Combining Equations 5.0.2, 5.0.3 and the Ohm's Law, $J = \sigma E$, where J is the electric current density, we have

$$v_d = \frac{\sigma E}{ne} = \frac{J}{ne} \tag{5.0.4}$$

Since the stable oxidation state of Al is three, the number of free electrons per m^3 of Al will be, $n = 3\frac{N_A \rho}{A} = 1.815 \times 10^{29}$ electrons/m^3 where N_A is the Avogadro's number, ρ is the density and A is the atomic weight. Assuming that, under normal operating conditions of integrated devices, current density can be as high as 10^{10} A/m^2 then, the drift velocity for Al will be calculated as 0.34 m/s. Conversion from SI units to atomic units gives the drift velocity as 1.5×10^{-7} a.u. In this study we have used 0.01 a.u. drift velocity. Although it is a rather high value compared to experiment, it was necessary to use such high values in order to observe an appreciable diffusion in a quite limited amount of real simulation time which is only 0.12 ps, where the simulation was carried out at almost half of the melting temperature.

The electromigration force was calculated by the use of a pseudopotential approach of Sorbello [14], where in this approach Heine-Abarenkov-Animalu form factors are used. The calculated electromigration force was added to the force obtained from the interatomic interactions at the corrector stage of the integration part. This added force can be thought as a perturbation which causes non-equilibrium conditions to be imposed on the system.

As mentioned earlier the electromigration force on atom j due to an applied electric field depends on the impurities and vacancies in the lattice, given by Equations 4.2.3 and 4.2.4.

$$F_j = -\frac{\partial U_j}{\partial R_j}$$

$$U_j = -\sum_{\substack{n \\ vacancies}} u_{jn} + \sum_{\substack{i \\ impurities}} u_{ji}$$

where u_{jm} has to be defined for all of the possible cases of pair interactions. The

possible pairs and their contribution to the potential in the calculation of electromigration force on the j^{th} atom is given in the Table 5.0.2.

In addition, the force distribution near a vacancy in an aluminum lattice and the electromigration force on various subtitutional atoms in an aluminum lattice was calculated separately from molecular dynamics simulations, actually before going into detailed MD studies. The aim was to determine the candidate substitutionals in aluminum that would have smaller electromigration force on them.

The evaluation of the force on the atoms due to neighbouring vacancies in the lattice is not an easy process compared to the force calculation due to the impurities in the lattice, since during the simulation, the positions of the vacancies should be traced. To determine the positions of the vacancies in each MD step, a subprogram was developed and this routine was called before each electromigration force calculation. The routine works in such a manner that the vacancies can be only located at perfect lattice sites. For each perfect lattice position therefore a sphere was constructed and it is controlled if there is any nearby atom located in this sphere. All of the atomic positions and their image positions in the MD cell is controlled for one perfect lattice position. The radius of the sphere is determined by several trials that give the correct vacancy number, which is introduced at the initialization part of the program. 1.145 Å of radius is used in this study compared to 1.431Å of half of the nearest neighbor distance in FCC aluminum.

In order to investigate the diffusion behavior of a material by molecular dynamics method, diffusion coefficient can be calculated by mean square displacement (MSD) (see Equation 2.1.34) or by velocity autocorrelation function (VAF) (see Equation 2.1.36). In non-equilibrium molecular dynamics however, Equation 2.2.11 can be used. In this study there is an applied electric field on the system, and this causes an electromigration force exerted on the atoms. This imposes the non-equilibrium conditions, thus excludes the use of MSD or VAF, since in non-equilibrium systems, approaches of statistical mechanics could not be used. Moreover, the treatment in NEMD can not differentiate the atomic species.

The calculation of the diffusion coefficient via EMD or NEMD approaches seems not to be possible under the conditions of the simulation in this study. In that case nothing but the basic definition of atomic transport is left, which is atomic jump frequency and it is defined as

Table 5.0.2: Possible pairs for calculation of the force on atom j.

type of j^{th} **atom**	**type of** m	$\mathbf{u}_{jm}$, for $m \leq N$
host atom	host atom	0
host atom	impurity	$= u_{jm} + \{ - \frac{\Omega^2 m k_F}{4\pi^3\hbar^2} \left(\frac{v_d}{v_F}\right) \cos(v_d, R_j - R_m)$ $\int_0^{2k_F} W_j(q) \left[W_m(q) - W_j(q)\right] q^2 j_1(q \left\|R_j - R_m\right\|) dq\}$
host atom	vacancy	$= u_{jm} - \{ - \frac{\Omega^2 m k_F}{4\pi^3\hbar^2} \left(\frac{v_d}{v_F}\right) \cos(v_d, R_j - R_m)$ $\int_0^{2k_F} W_j(q)^2 q^2 j_1(q \left\|R_j - R_m\right\|) dq\}$
impurity	host atom	0
impurity	impurity	$= u_{jm} + \{ - \frac{\Omega^2 m k_F}{12\pi^3\hbar^2} \left(\frac{v_d \cdot R_j}{v_F}\right)$ $\int_0^{2k_F} W_j(q) \left[W_j(q) - W_m(q)\right] q^3 dq\}$ where m is $+\{ - \frac{\Omega^2 m k_F}{4\pi^3\hbar^2} \left(\frac{v_d}{v_F}\right) \cos(v_d, R_j - R_m)$ host atom $\int_0^{2k_F} W_j(q) \left[W_j(q) - W_m(q)\right] q^2 j_1(q \left\|R_j - R_m\right\|) dq\}$
impurity	vacancy	$= u_{jm} - \{ - \frac{\Omega^2 m k_F}{4\pi^3\hbar^2} \left(\frac{v_d}{v_F}\right) \cos(v_d, R_j - R_m)$ $\int_0^{2k_F} W_j(q) W_m(q) q^2 j_1(q \left\|R_j - R_m\right\|) dq\}$ $+\{ - \frac{\Omega^2 m k_F}{12\pi^3\hbar^2} \left(\frac{v_d \cdot R_j}{v_F}\right)$ where m is host atom $\int_0^{2k_F} W_j(q) \left[W_j(q) - W_m(q)\right] q^3 dq\}$

$$\Gamma = \left\langle \frac{n}{t} \right\rangle$$

where Γ is the jump frequency of atoms, n is the number of jumps associated by atoms during a time t. It is assumed that the successful jump counts if the atom moved by a distance not less than the nearest neighbor distance in FCC.

Chapter 6

Results and Discussion

6.1 Critical Potential Evaluation for Aluminum

For the evaluation of atomic interactions in the molecular dynamics simulations, pair potentials were constructed using Heine-Abarenkov-Animalu model pseudopotentials. Various screening functions, which are Hubbard, Kleinman, Langreth, Geldart-Vosko, Ichimaru-Utsumi, Moronoi and Vashishta-Singwi screening functions, were used in the potential production for aluminum. The obtained potentials were given in Figure 6.1.1.

The potentials used in the simulations should be tested in order to be confident about their applicability in simulations. For this purpose the validity of the potentials produced with different screening functions were tested by GULP [82] software and some of the properties, which were lattice parameter, lattice energy and elastic constants of aluminum were calculated and compared with the experimental ones, as given in Table 6.1.1. First and one of the most important expectation from the potential is its capability to predict the right ground state, which is FCC for Al. As can be seen from Table 6.1.1 for all screening functions and from Figure 6.1.2 in detail for Geldart-Vosko screening, the right ground state FCC was predicted. Then the analysis of the lattice parameter, elastic constants, etc. becomes important. The best prediction of the lattice parameter was obtained with Geldart-Vosko screening, see Table 6.1.1. It predicts the lattice parameter with 2.96% error. Therefore, Geldart-Vosko was chosen as the screening function for the production model pair potentials using Heine-Abarenkov-Animalu pseudopotential form factors.

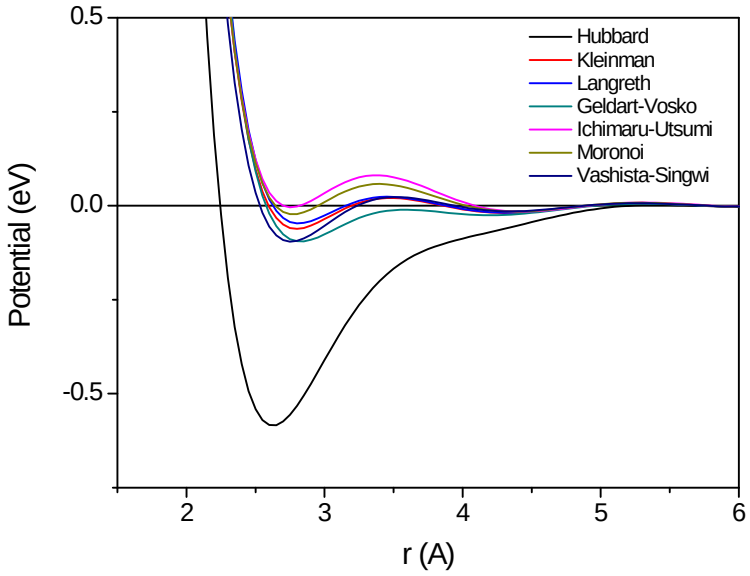

Figure 6.1.1: Heine-Abarenkov-Animalu potentials with various screening functions.

In addition self-diffusion coefficients of aluminum were calculated at several temperatures using the mean square displacement (MSD) equation (Equation 2.1.34) with the simulation of the system under equilibrium conditions. In order to calculate the temperature dependence of the diffusion coefficient ($D(T)$), molecular dynamics simulations were carried out at 800 K, 1000 K and 1200 K. For each temperature mean square displacement was sampled over 150000 time steps which corresponds to 14.73 ps. Mean-square displacements obtained at different temperatures were given in Figure 6.1.3. From the measured slopes of the MSD plots the diffusivity of aluminum were calculated to be 1.7×10^{-11}, 5.8×10^{-11}, 5.64×10^{-10} m^2/s at 800 K, 1000 K, 1200 K respectively.

Fitting this data to the well known Arrhenius equation,

$$D = D_0 \exp(\frac{-Q}{k_b T})$$

gives D_0, temperature independent constant and Q, the activation energy for diffusion. The lnD vs. $1/k_bT$ plot was given in Figure 6.1.4. From the slope of this curve, activation energy was calculated as $Q = 67.6$ kJ/mole and $D_0 = 3.50 \times 10^{-7}$ m^2/s, compared to the experimental values [87] 124 kJ/mole and 1.37×10^{-5} m^2/s respectively. The activation energy predicted was almost half of the experimental

Table 6.1.1: Properties of Aluminum obtained from various screening functions on Heine-Abarenkov-Animalu model potentials.

Screening Function	Crystal Structure	Lattice Parameter (Å)	Lattice Energy (eV)	C_{11} (10^{11}dyn/cm^2)	C_{12} (10^{11}dyn/cm^2)
Experimental	fcc	4.05 [84]	-3.3983 [85]	10.649 [86]	5.988 [86]
Hubbard	fcc	3.5880	-17.9828	43.134	28.585
	bcc	2.8124	-9.0048	48.784	37.126
Kleinmann	fcc	3.9137	-1.8017	24.188	8.899
	bcc	3.1221	-1.0708	3.754	10.711
Langreth	fcc	3.9321	-1.4072	22.832	8.166
	bcc	3.1471	-0.6949	0.745	0.248
Geldart - Vosko	fcc	3.9301	-2.8189	22.396	8.704
	bcc	3.1408	-2.1688	12.902	4.301
Ichimaru - Utsumi	fcc	3.8982	-0.1413	27.699	9.342
	bcc	3.0909	0.7371	4.771	12.149
Moronoi	fcc	3.9025	-0.6469	26.041	8.988
	bcc	3.1078	0.1744	3.451	11.154
Vashishta - Singwi	fcc	3.8762	-2.5089	26.965	10.669
	bcc	3.0747	-1.7758	7.546	13.321

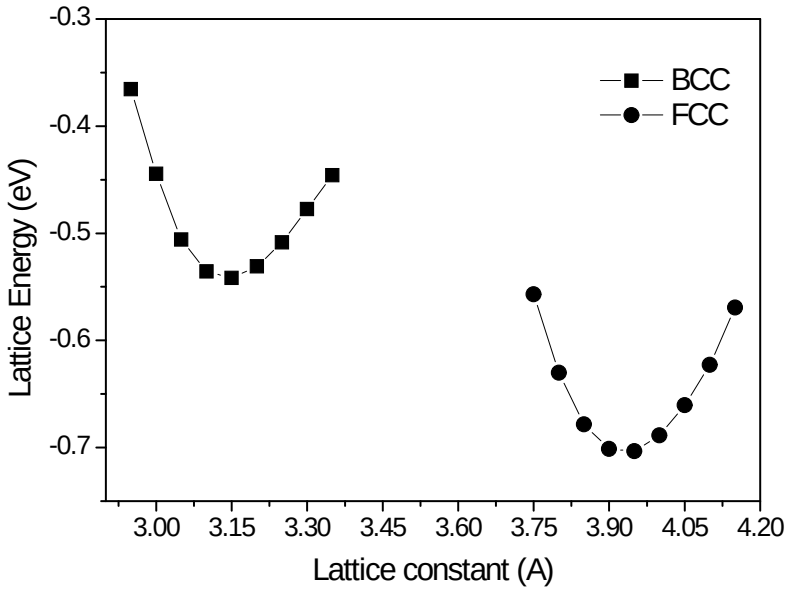

Figure 6.1.2: Ground state analysis for Heine-Abarenkov-Animalu model potentials.

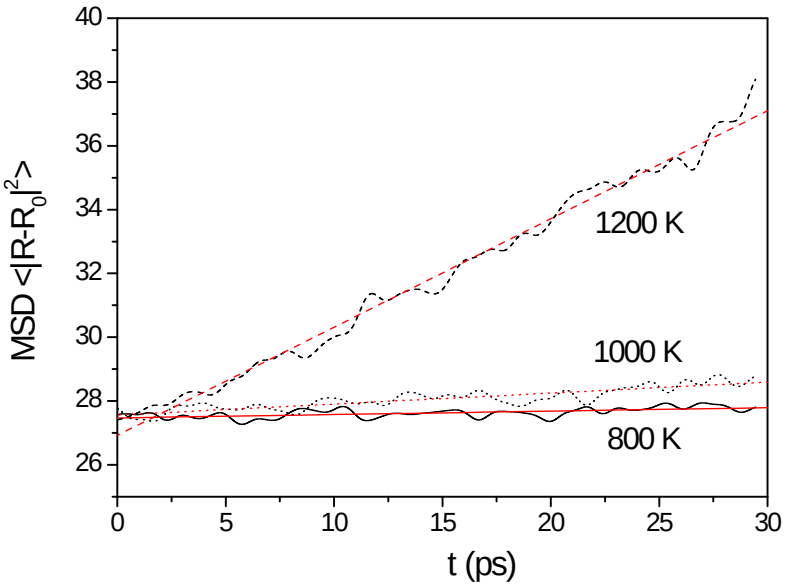

Figure 6.1.3: Mean square displacement for different temperatures.

value. This may be due to the very high concentration of vacancies in the system.

6.2 Electromigration Force Calculation Results

The electron scattering forces on several substitutional impurities were calculated using Equation 4.2.6 in aluminum with Heine-Abarenkov-Animalu model pseudopotentials and were given in Table 6.2.1. In addition, the average force, Equation 4.2.7, for diffusion by vacancy mechanism for aluminum was calculated as 0.0633 eV/A. There seems to be a relation between electron scattering force and the number of valence electrons of the substitutional atoms. Li, Cu, Na, Pd has the least scattering force acting on them. From Table 6.2.1, it can also be seen that several substitutional atoms has a negative scattering force, opposite to the electron drift.

The distribution of the electron scattering force nearby a vacancy and substitutional impurities were also calculated. The force distribution nearby a vacancy in the (111) plane on the FCC lattice can be seen in Figure 6.2.1. In the figure, vacancy is placed at the middle, which is denoted by a square. The arrows indicate the direction

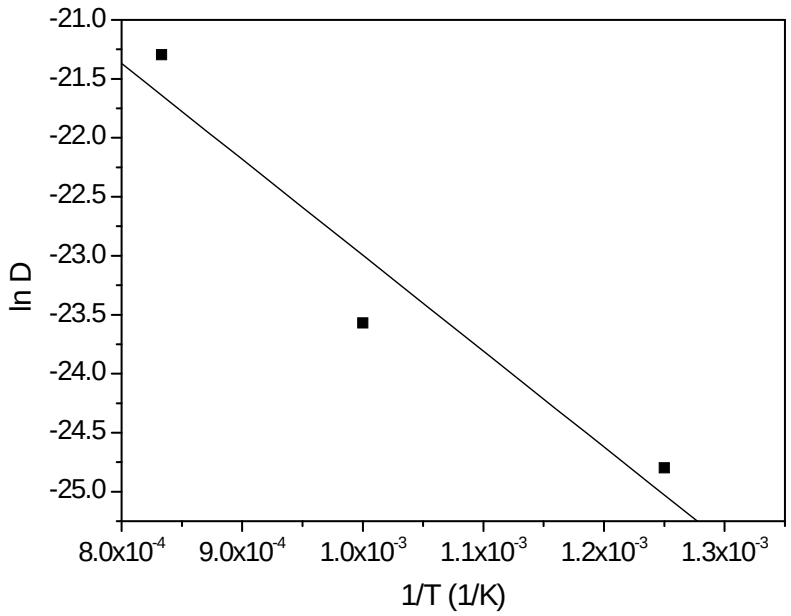

Figure 6.1.4: Arrhenius plot for the diffusion coefficient calculated from the mean square displacements.

of the force on each atom and the numbers give the scaled magnitude of the force. It can be seen that the magnitude of force on the atoms nearby the vacancy is quite high. The largest force on the atoms are in the direction of the electron scattering force.

Electromigration force distribution nearby the vacancy with a substitutional atom were given in Figures 6.2.2, 6.2.3, 6.2.4, 6.2.5 and 6.2.6 for Cu, Mg, Mn, Sn and Ti respectively. The distribution of the electromigration forces in substitutional impurity systems varied depending on the nature of the impurity elements. The addition of Sn and Ti causes a negative force, in the opposite direction of the drift velocity, on the host atoms nearby an impurity, while the addition of Cu, Mg, Mn, do not alter the direction of the forces as in the vacancy case. The addition of a substitutional element mainly influences the state of the force on the host atoms which are nearest neighbors to both the impurity and the vacancy. In addition, the forces calculated on Ti and Sn atoms are much higher in magnitude than the other alloying elements, which were aligned in the direction of the drift velocity. However the force calculated on Cu was very small and aligned in the opposite direction of the drift velocity.

Table 6.2.1: Electron scattering forces on elements that are added subtitutionally to the aluminum with a drift velocity of 0.01 a.u. All forces are in eV/A.

Element	F_{sub}	Element	F_{sub}	Element	F_{sub}	Element	F_{sub}
Li	0.00895	V	0.2944	Rb	0.196	Sn	0.0304
Be	0.0159	Cr	0.0362	Y	0.161	Sb	0.0901
B	0.1162	Mn	-0.0173	Zr	0.197	Te	0.1678
C	0.3169	Fe	0.0233	Nb	0.2128	Cs	0.2548
Na	0.0033	Co	-0.0143	Mo	0.2468	Ba	0.8022
Mg	-0.01024	Ni	-0.0144	Tc	0.1557	La	0.1086
Si	0.04903	Cu	-0.0018	Ru	0.1899	Hf	0.1413
P	0.1727	Zn	-0.0122	Rh	0.1518	Ta	0.1999
K	0.0938	Ga	0.0123	Pd	0.0024	W	0.2184
Ca	0.1141	Ge	0.0522	Ag	0.0152	Au	0.0941
Sc	0.0166	As	0.1535	Cd	0.0098	Pb	0.0425
Ti	0.1028	Se	0.2786	In	0.0137		

6.3 Non-Equilibrium Molecular Dynamics Results

The addition of electromigration force to the force calculated from the atomic interactions was done on the corrector stage of the integration part in the molecular dynamics program. The application of an external force on the atoms imposes non-equilibrium conditions, where the approaches of the statistical mechanics can no longer be valid and the determination of diffusion coefficient by the use of mean square displacement or velocity autocorrelation function could not be carried out. Additionally, diffusion behavior can not be understood by the calculation of the diffusion coefficient, defined for the systems, given by Equation 2.2.11, since it will be analogous to the diffusion coefficient calculated by equilibrium molecular dynamics. As a consequence, to characterize the diffusion behavior, the description of the jump frequency was used, in which for each Al atom in the box, the elapsed time for an atom to jump to another lattice position and the number of jumps were calculated. A jump was characterized by the movement of an atom by the nearest neighbor distance in FCC, which is 2.86 Å for aluminum. The jump frequency was then obtained by averaging the number of jumps per second over all of the Al atoms in the system. The calculated jump frequencies of aluminum atoms for alloy systems investigated were given in Table 6.3.1.

As can be seen from Table 6.3.1, with the increase in temperature, except for Ti system, jump frequencies were also increased as expected. In Ti alloyed system, there is a small decrease in the jump frequency going from 1200 K to 1400 K. This may

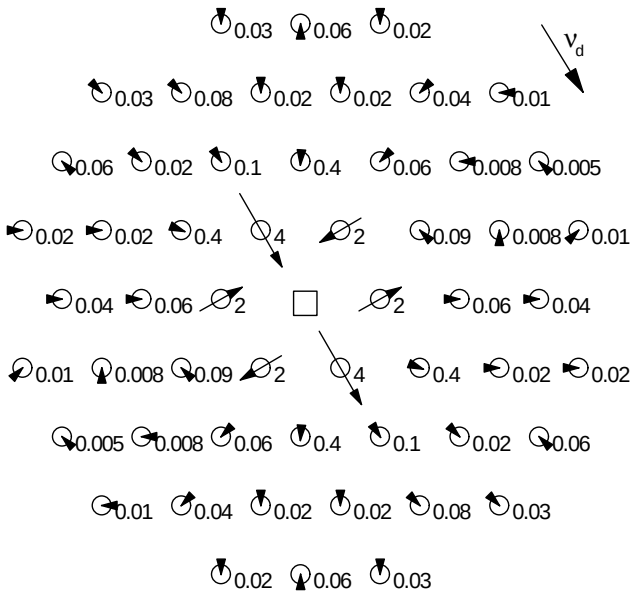

Figure 6.2.1: The force distribution near a vacancy in (111) plane for Aluminum

be due to some inefficiency in the potential. In addition, for all impurities added, a decrease in the jump frequency of Al was observed. However the amount of the decrease changes from impurity to impurity. At 1000 K, the most effective additions are Cu, Mn and Sn. Other elements Mg and Ti were not so effective. For Cu, Mn and Sn, the jump frequency at 1000 K was decreased respectively to 3.952×10^{-10}, to 5.254×10^{-10} and to 4.863×10^{-10} jumps per second, compared to jump frequency in pure Al, from 12.236×10^{-10} jumps/sec. As temperature increases, the above trend continues to hold except for Mn, where at 1200 K for example, the jump frequency of Al in the Al-Mn alloy increased to 21.518×10^{-10} beyond the value of 18.58×10^{-10} in the pure Al. As temperature rises to 1400 K, the system was so agitated that the effect of the alloying elements becomes less significant.

In order to understand the diffusion behavior of aluminum with the addition of alloying elements under the electromigration force, the force distribution generated by the alloying element and the character of the interatomic potential between aluminum and the alloying element was considered simultaneously. For this purpose, furthermore, equilibrium MD simulations of pure Al and Cu added alloy were carried

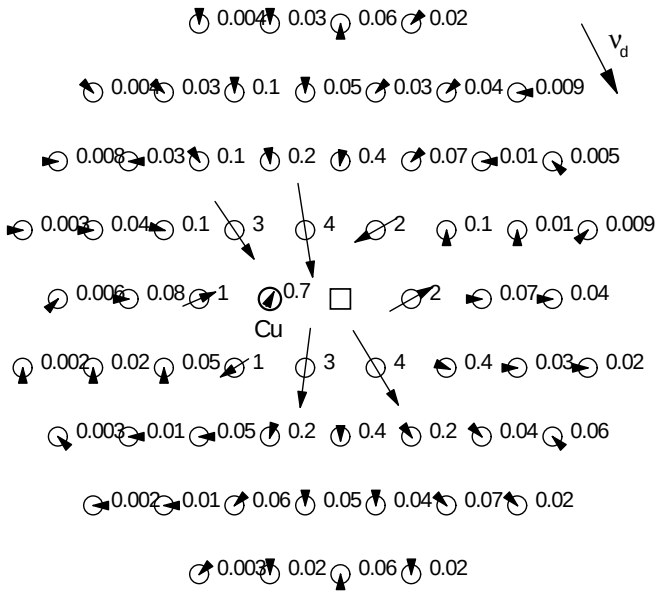

Figure 6.2.2: Electromigration force distribution nearby a Cu substitutional on (111) plane.

out in exactly the same way as done before, but this time the electromigration force was turned off. The calculated jump frequencies at 1000 K were given in Table 6.3.2.

It is interesting to note that, the jump frequencies of Al decreases as electromigration force was turned on. As already mentioned by Tu [83], an experimentally observed similar behavior could not be explained.

In Figures 6.3.1 and 6.3.2, the calculated force distributions, due to atomic interactions only, around a vacancy and vacancy and Cu substitutional impurity couple in the (111) plane of the FCC lattice were given, respectively. In these figures nearest neighbors were considered only and the arrows show the direction and the numbers denote the scaled magnitude of the projected forces on the (111) plane. In three dimensions actually, for the case of vacancy for example, all forces should be equal to each other because of the symmetry. Since in three dimensions the visualization of the picture is not so easy, the two dimensional picture was preferred.

As can be seen in these figures because of the missing chemical bond, all atoms around the vacancy were pulled away from the vacancy. With the inclusion of Cu,

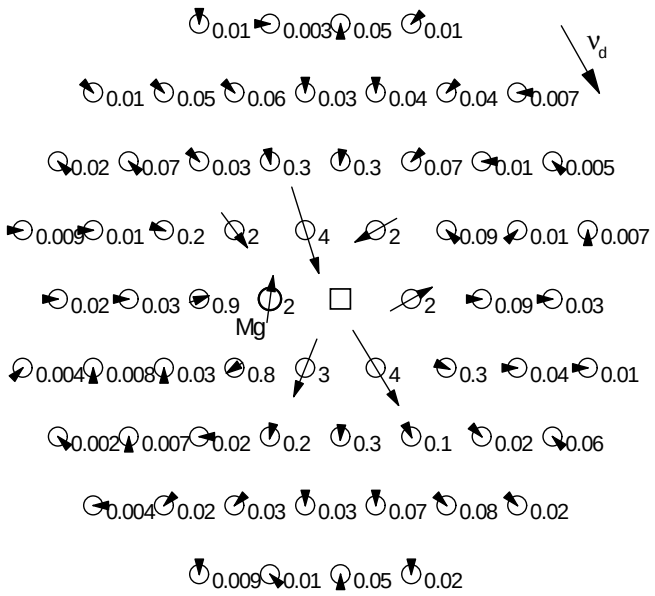

Figure 6.2.3: Electromigration force distribution nearby a Mg substitutional on (111) plane.

however, the forces on Al atoms were modified such that they are now stronger. This means that, for the exchange of a vacancy and a nearest neighbor Al atom, larger forces have to be overcome and thus a slower diffusion kinetics should be expected. This may explain why in Cu substituted alloys jump frequency decreases. When we apply the electric field, in addition to the forces due to interatomic interactions, electromigration forces were developed as well. The magnitudes of these forces are such that, interatomic forces are several orders of magnitude larger than electromigration forces, therefore the scaling in Figures 6.2.1 - 6.2.6 and in Figures 6.3.1 - 6.3.2 were not the same. The inclusion of the electromigration force, in both cases of pure and Cu added, resulted in the weakening of some and strengthening of some other forces. In the case of pure Al, however, the additional electromigration forces on the neighboring Al atoms tend to cancel each other, see Figure 6.2.1, and should result in no change in the jumping frequencies, where in Table 6.3.2, there is only a slight difference in the jumping frequencies. For the case of Cu added alloy, the picture is somewhat different, see Figure 6.2.2, such that there is slight strengthening of the net force due to electromigration. This means that one should expect a little more

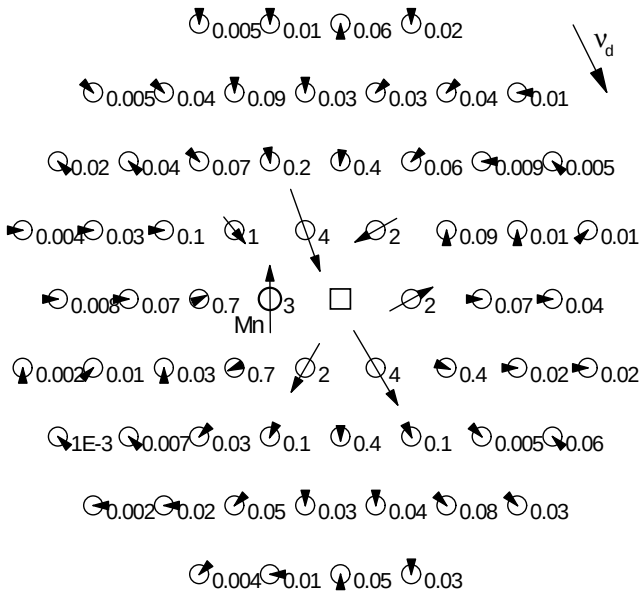

Figure 6.2.4: Electromigration force distribution nearby a Mn substitutional on (111) plane.

difficult diffusion thus slower kinetics under electromigration conditions. Another concern may be, the forces developed on the impurities themselves. For the cases of Cu and Ti added alloys similar force picture was obtained at the surrounding Al atoms, thus one should expect similar diffusion behavior. However they are totally different. This difference may be because of the difference of the forces on the impurities. Electromigration force on Cu, for example, was calculated to be so small, whereas for Ti it is very large, meaning that Ti itself experiences a large disturbance due to electron scattering. This may cause agility in the surroundings as well. Therefore compared to Cu, it may not cause a similar reduction in the diffusion kinetics of the Al atoms.

Therefore diffusion of the host aluminum atoms seems to depend in a complicated way on how electromigration and interatomic forces are developed. The method presented in this study did not contain full physics, therefore any further attempt to answer how and why, would be nothing but speculation. The answers can only be given by ab initio methods where quantum mechanical description of forces

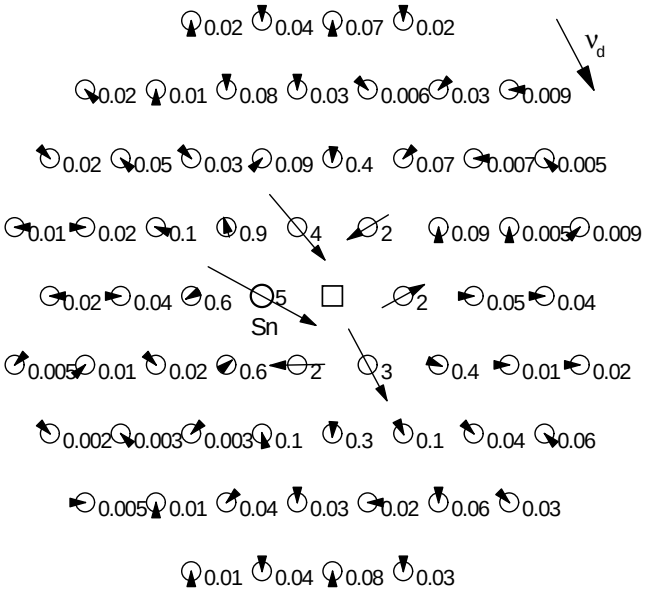

Figure 6.2.5: Electromigration force distribution nearby a Sn substitutional on (111) plane.

and response functions to electric field have to be dealt with.

The direct result of this study was the calculated jump frequencies. The ability of the developed method to predict the experimental situation in Cu added Al alloys, make the results obtained for Sn and Mn prominent, such that these elements would also be effective in improving the properties of Al interconnects.

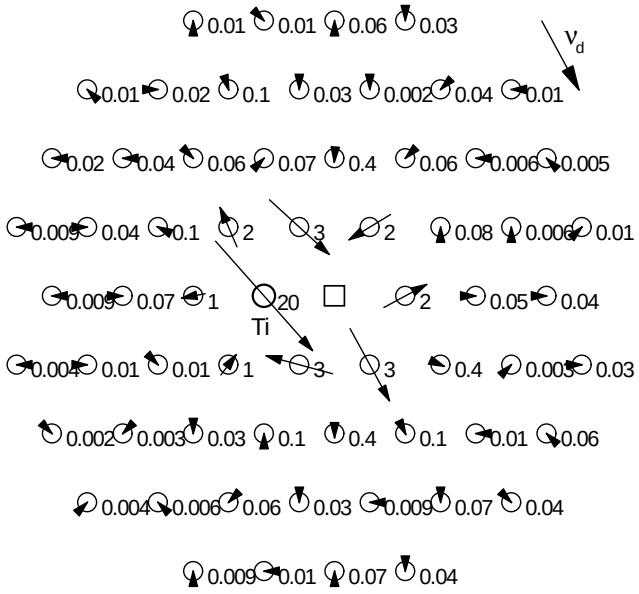

Figure 6.2.6: Electromigration force distribution nearby a Ti substitutional on (111) plane.

Table 6.3.1: Calculated jump frequencies ($\times 10^{-10}$jumps/sec) of aluminum atoms in given systems at different temperatures.

System	**1000 K**	**1200 K**	**1400 K**
Al	12.236	18.580	23.693
Al-1.125at% Cu	3.952	10.169	24.031
Al-1.125at% Mg	9.125	16.012	30.825
Al-1.125at% Mn	5.254	21.518	29.960
Al-1.125at% Sn	4.863	10.433	25.225
Al-1.125at% Ti	11.049	21.198	19.494

Table 6.3.2: Calculated jump frequecies ($\times 10^{-10}$jumps/sec) of Al at 1000 K with and without the electromigration force.

System	**EM force off**	**EM force on**
Al	12.975	12.236
Al-1.125at% Cu	4.838	3.952

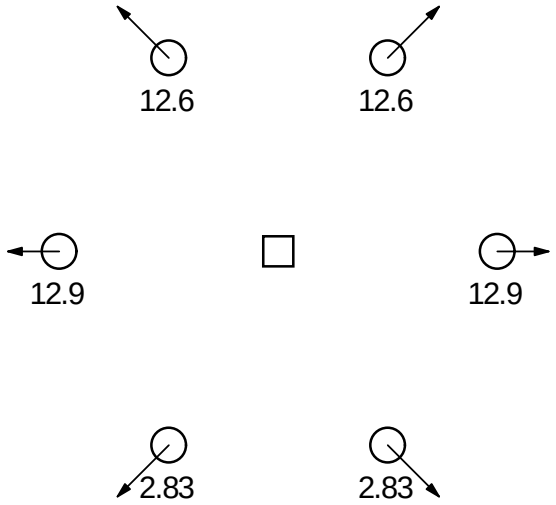

Figure 6.3.1: Force distribution nearby a vacancy due to interatomic interactions in (111) plane.

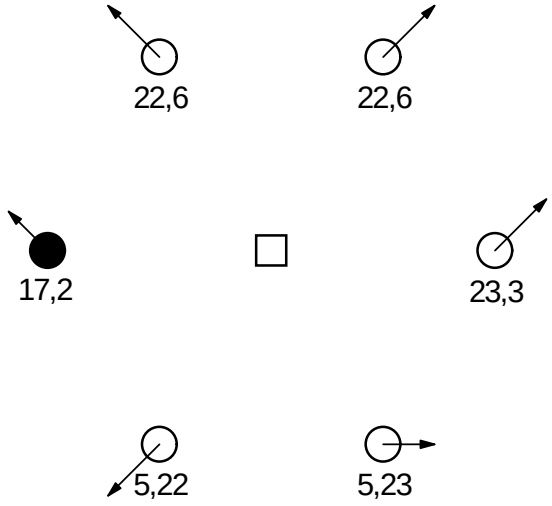

Figure 6.3.2: Force distribution nearby a vacancy - Cu substitutional impurity couple due to interatomic interactions in (111) plane.

Chapter 7

Conclusion

In the present study, the effect of alloying aluminum on the diffusion behavior under electromigration force was investigated using non-equilibrium molecular dynamics simulations. The electromigration force was calculated by a pseudopotential formalism and the calculated force was added to the interatomic force in the corrector stage of the molecular dynamics. It is shown in this study that the developed procedure can be used to understand the electromigration processes occurring in metallic interconnect alloys.

Heine-Abarenkov-Animalu model potentials were used in the atomic interactions. The potentials were tested for the aluminum such that some of the ground state properties were calculated and compared with the experimental values. The calculated properties were found to be close to the experimental values where, lattice parameter was obtained with 2.96% error and self-diffusion activation energy for aluminum was found to be 67.6 kJ/mole.

Various alloying elements were added to the aluminum and the corresponding electromigration force on the alloying elements were determined. It has been found that the force on Cu, Li, Na and Pd is smaller compared to other elements. In transition metals there is a direct relation between the electron scattering force and the chemical valence of the element.

The change in the jump frequencies of aluminum atoms were reported at different temperatures by addition of Cu, Mg, Mn, Sn and Ti to the aluminum. It was found that, Cu, Mn and Sn reduces the kinetics of the jumps of aluminum atoms. Therefore, it is suggested that Cu, Mn and Sn can be used as an alloying element to improve the resistance of aluminum to electromigration. Thus longer failure times and an increase in the reliability of the interconnects can be achieved.

References

[1] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21:1087–92, 1953.

[2] B. J. Alder and T.E. Wainwright. Phase transition for a hard sphere system. *Journal of Chemical Physics*, 27:1208–9, 1957.

[3] B. J. Alder and T. E. Wainwright. Studies in molecular dynamics. *Journal of Chemical Physics*, 31:459–66, 1959.

[4] A. Rahman. Correlations in the motion of atoms in liquid argon. *Physical Review*, 136:A405, 1964.

[5] G. Sutmann. Classical molecular dynamics. *Quantum Simulations of Complex Many-Body Systems: From Theory to Algorithms, Lecture Notes*, 10:211–254, 2002.

[6] A. Davutoğlu. *Molecular Dynamics Studies of Diffusion Behaviour in Aluminum-Copper Metals and Alloys*. PhD thesis, Middle East Technical University, September 2003.

[7] K. P. Travis. *Simulation of the Liquid State*, chapter 6, pages 217–270. Royal Society of Chemistry, 2004.

[8] W. G. Hoover and W. T. Ashurst. *Theoretical Chemistry: Advances and Perspectives*, volume 1, chapter Nonequilibrium molecular dynamics, pages 1–51. Academic Press, New York, 1975.

[9] D. J. Evans and G. P. Morris. *Statistical Mechanics of Nonequilibrium Liquids*. Academic Press, New York, 1990.

[10] J. R. Lloyd. Electromigration for designers: An introduction for the non specialist. Technical report, IBM TJ Watson Research Center, 2002.

[11] D. R. Fridline and A. F. Bower. Influence of anisotropic surface diffusivity on electromigration induced void migration and evolution. *Journal of Applied Physics*, 85:3168, 1999.

[12] J. P. Dekker, C. A. Volkert, E. Arzt, and P. Gumbsch. Alloying effects on electromigration mass transport. *Physical Review Letters*, 87(3):035901, 2001.

[13] T. A. Ogurtani and E. E. Oren. Computer simulation of void growth dynamics under the action of electromigration and capillary forces in narrow thin interconnects. *Journal of Applied Physics*, 90(3):1564, 2001.

[14] R. S. Sorbello. A pseudopotential based theory of the driving forces for electromigration in metals. *Journal of Physics and Chemistry of Solids*, 34:937–950, 1972.

[15] R. S. Sorbello. Atomic configuration effects in electromigration. *Journal of Physics Chemistry of Solids*, 42:309–316, 1980.

[16] P. Kumar and R. S. Sorbello. Linear response theory of the driving forces for electromigration. *Thin Solid Films*, 25:25–35, 1975.

[17] T. Ohkubo, Y.Hirotsu, and K.Nikawa. Molecular dynamics simulation of electromigration in nano-sized metal lines. *Materials Transactions, JIM*, 37(3):454–457, 1996.

[18] T. Shinzawa and T. Ohta. Molecular dynamics simulation of al grain boundary diffusion for electromigration failure analysis. In *Proceedings of the IEEE 1998 International Interconnect Technology Conference*, 1998.

[19] D. Maroudas and M. R. Gungor. Continuum and atomistic modeling of electromechanically-induced failure of ductile metallic thin films. *Computational materials science*, 23:242–249, 2002.

[20] J. B. Gibson, A. N. Goland, M. Milgram, and G. H. Vineyard. Dynamics of radiation damage. *Physical Review*, 120:1229–1253, 1960.

[21] L. Verlet. Computer 'experiments' on classical fluids. i. thermodynamical properties of lennard-jones molecules. *Physical Review*, 159:98–103, 1967.

[22] L. Verlet. Computer 'experiments' on classical fluids. II. equilibrium correlation functions. *Physical Review*, 165:201–214, 1968.

[23] F. Ercolessi. A molecular dynamics primer. http://www.sissa.it/furio/md, 2006.

[24] A. Münster. *Statistical Thermodynamics*. Springer/Academic Press, Berlin/New York, 1969.

[25] I. M. Torrens. *Interatomic Potentials*. Academic Press, New York, 1972.

[26] M. P. Allen and D. J. Tildesley. *Computer Simulation of Liquids*. Oxford University Press, New York, 1987.

[27] M. P. Allen. Introduction to molecular dynamics simulation. *Computational Soft Matter: From Synthetic Polymers to Proteins, Lecture Notes*, 23:1–28, 2004.

[28] R. W. Hockney and J. W. Eastwood. *Computer Simulations Using Particles*. Adam Hilger, Bristol, 1988.

[29] W. C. Swope, H. C. Andersen, P. H. Berens, and K. R. Wilson. 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*, 76:637–649, 1982.

[30] C. W. Gear. The numerical integration of ordinary differential equations of various orders. Technical Report Report ANL 7126, Argonne National Laboratory, 1966.

[31] C. W. Gear. *Numerical Initial Value Problems in Ordinary Differential Equations*. Prentice-Hall, Englewood Cliffs, NJ, 1971.

[32] M. Born and Th. Von Karman. über schwingungen in raumgittern. *Physik. Z.*, 13:297–309, 1912.

[33] D. W. Heermann. *Computer Simulation Methods in Theoretical Physics*. Springer-Verlag, Germany, 1990.

[34] L. V. Woodcock. Isothermal molecular dynamics calculation for liquid salt. *Chemical Physics Letters*, 10:257–261, 1971.

[35] H. J. C. Berendsen, J. P. M. Postma, W. F. Van Gusteren, A. DiNola, and J. R. Haak. Molecular dynamics with coupling to an external bath. *Journal of Chemical Physics*, 81(8):3684–3690, 1984.

[36] S. Nose. A molecular dynamics method for simulations in the canonical ensemble. *Molecular Physics*, 52:255–268, 1984.

[37] W. H. Hoover. Canonical dynamics: Equilibrium phase-space distributions. *Physical Review A*, 31(3):1695–1697, 1985.

[38] W. G. Hoover. Constant pressure equations of motion. *Physical Review A*, 34:2499–2500, 1986.

[39] S. Nose. An extension of the canonical ensemble molecular dynamics method. *Molecular Physics*, 57:187–191, 1986.

[40] R. Windiks. Thermostatting in molecular dynamics simulations. http://homepage.swissonline.ch/windiks/documents/thermostats.pdf, 2006.

[41] H. C. Andersen. Molecular dynamics simulations at constant pressure and/or temperature. *Journal of Chemical Physics*, 72:2384–2393, 1980.

[42] J. M. Haile and H. W. Graben. Molecular dynamics simulations extended to various ensembles. i. equilibrium properties in the isoenthalpic-isobaric ensemble. *Journal of Chemical Physics*, 73:2412–2419, 1980.

[43] A. Hinchliffe. *Chemical Modelling from Atoms to Liquids*. John Wiley and Sons Ltd, 1999.

[44] H. J. C. Berendsen and W. F. Van Gusteren. Practical algorithms for dynamic simulations. molecular dynamics simulation of statistical mechanical systems. In *Proceedings of the Enrico Fermi Summer School*, pages 43–65, Bologna, 1986. Soc. Italiana di Fisica.

[45] A. R. Leach. *Molecular Modelling Principles and Applications*. Addison Wesley Longman Limited, 1996.

[46] D. J. Evans and G. P. Morriss. Non-newtonian molela dynamics. *Computer Physics Reports*, 1(6):297–343, 1984.

[47] J. J. Erpenbeck and W. W. Wood. Molecular dynamics techniques for hard core particles. In B. J. Berne, editor, *Statistical Mechanics B. Modern Theoretical Chemistry*, pages 1–40. Plenum, New York, 1977.

[48] A. A. Katsnelson, V. S. Stepanyuk, A. I. Szasz, and O. V. Farberovich. *Computational Methods in Condensed Matter: Electronic Structure*. AIP, New York, 1992.

[49] D. R. Hamann, M. Schlüter, and C. Chiang. Norm-conserving pseudopotentials. *Physical Review Letters*, 43:1494–1497, 1979.

[50] D. R. Hamann. Generalized norm-conserving pseudopotentials. *Physical Review B*, 40:2980–2987, 1989.

[51] A. M. Rappe, K. M. Rabe, E. Kaxiras, and J. D. Joannopoulos. Optimized pseudopotentials. *Physical Review B*, 41:1227–1230, 1990.

[52] N. Troulier and J. L. Martins. Efficient pseudopotentials for plane-wave calculations. *Physical Review B*, 43:1993–2006, 1991.

[53] D. Vanderbilt. Optimally smooth norm-conserving pseudopotentials. *Physical Review B*, 32:8412–8415, 1985.

[54] M. Fuchs and Scheffler. Ab initio pseudopotentials for electronic structure calculations of poly-atomic systems using density-functional theory. *Computational Physics Communications*, 119:67, 1999.

[55] N. W. Ashcroft. Electron-ion pseudopotentials in metals. *Physical Letters*, 23:48, 1966.

[56] V. Heine and I. V. Abarenkov. New method for electronic structure of metals. *Philosophical Magazine*, 9:451, 1964.

[57] I. V. Abarenkov and V. Heine. Model potential for positive ions. *Philosophical Magazine*, 12:529, 1965.

[58] A. O. E. Animalu. Electronic structure of transition metals. II. phonon spectra. *Physical Review B*, 8:3542, 1973.

[59] W. A. Harrison and J. M. Wills. Interionic interactions in simple metals. *Physical Review B*, 25:5007, 1982.

[60] V. N. Antonov, V. Y. Milman, V. V. Nemoshkalenko, and A. V. Zhalkotitarenko. Lattice-dynamics of fcc transition-metals-a pseudopotential approach. *Zeitschrift fur Physik B - Condensed Matter*, 79:223, 1990.

[61] C. Fiolhais, J. P. Perdew, S. Q. Armster, J. M. MacLaren, and M. Brajczewska. Dominant density parameters and local pseudopotentials for simple metals. *Physical Review B*, 51:14001, 1995.

[62] J. L. Bretonnet, G. M. Bhuiyan, and M. Silbert. Gibbs-bogoliubov variational scheme calculations for the liquid structure of 3d transition metals. *Jornal of Physics Condensed Matter*, 4:5359, 1992.

[63] J. L. Bretonnet and M. Silbert. Interionic interactions in transition-metals-application to vanadium. *Physics and Chemistry of Liquids*, 24:169, 1992.

[64] M. Hasegawa, K. Hoshino, and M. Watabe. A new simple pseudopotential with applications to liquid-metal structure factor. *Journal of Non-Crystalline Solids*, 117/118:300, 1990.

[65] A. O. E. Animalu and V. Heine. The screened model potential for 25 elements. *Philosophical Magazine*, 12:1249, 1965.

[66] A. O. E. Animalu. The total electronic band structure energy for 29 elements. *Proceedings of the Royal Society of London Series A, Mathematical and Physical Sciences*, 294(1438):376–392, 1966.

[67] J. Lindhard. On the properties of a gas of charged particles. *Matematisk-fysiske meddelelser Kongelige Danske Videnskabernes Selskab*, 28(8), 1954.

[68] J. Hubbard. The description of collective motions in terms of many-body perturbation theory.2.the correlation energy of a free-electron gas. *Proceedings of the Royal Society of London Series A-mathematical and physical sciences*, A243:336, 1958.

[69] L. Kleinman. New approximation for screened exchange and the dielectric constant of metals. *Physical Review*, 160:585, 1967.

[70] D. J. W. Geldart and S. H. Vosko. Screening function of an interacting electron gas. *Canadian Journal of Physics*, 44:2137, 1966.

[71] D. C. Langreth. Approximate screening functions in metals. *Physical Review*, 181:753, 1969.

[72] S. Ichimaru and K. Utsumi. Analytic expression for the dielectric screening function of strongly coupled electron liquids at metallic and lower densities. *Physical Review B*, 24:7385, 1981.

[73] P. Vashista and K. S. Singwi. Electron correlations at metallic densities. *Physical Review B*, 6:875, 1972.

[74] S. Moronoi, D. M. Ceperley, and G. Senatore. Static response and local field factor of the electron gas. *Physical Review Letters*, 75:689, 1995.

[75] P. S. Ho and T. Kwok. Electromigration in metals. *Reports on Progress in Physics*, 52:301–348, 1989.

[76] R. S. Sorbello. Theory of electromigration. In *Solid State Physics*, volume 51, pages 159–231. Academic„ New York, 1997. Vol 51.

[77] V. B. Fiks. On the mechanism of the mobility of ions in metals. *Soviet Physics -Solid State*, 1:14, 1959.

[78] H. B. Huntington and A. R. Grone. Current induced marker motion in gold wires. *Journal of Physics and Chemistry of Solids*, 20:76, 1961.

[79] C. Bosvieux and J. Friedel. Sur leelectrolyse des alliages metalliques. *Journal of Physics and Chemistry of Solids*, 20:76, 1961.

[80] T. N. Todorov, J. Hoekstra, and A. P. Sutton. Current-induced forces in atomic-scale conductors. *Philosophical Magazine B*, 80(3):421–455, 2000.

[81] J. P. Dekker and A. Lodder. Calculated electromigration wind force in face-centered-cubic and body-centered-cubic metals. *Journal of Applied Physics*, 4:1958, 1998.

[82] J. D. Gale and A. L. Rohl. The general utility lattice program. *Molecular Simulation*, 29:291, 2003.

[83] K. N. Tu. Recent advances on electromigration in very-large-scale-integration of interconnects. *Journal of applied physics*, 94(9):5451–5473, 2003.

[84] J. Bandyopadhyay and K. P. Gupta. Low temperature lattice parameters of al and al-zn alloys and grüneisen parameter of al. *Cryogenics, London*, 18:54–55, 1978.

[85] C. Kittel. *Introduction to Solid State Physics*. John Wiley and Sons, New York, 1986.

[86] G. Simmons and H. Wang. *Single-Crystal Elastic Constants and Calculated Aggregate Properties: A Handbook*. MIT Press, MA, 1971.

[87] S. Dais, R. Messer, and A. Seeger. Nuclear-magnetic-resonance study of self-diffusion in aluminum. *Materials Science Forum*, 15-18:419–424, 1987.

Appendix A

List of Computer Program

```
!! Common declerations
module commondefs
save
type atom
  real x0, y0, z0, x, y, z, fx, fy, fz, elfx, elfy, elfz, vx, vy, vz
  real ax, ay, az, bx, by, bz, cx, cy, cz, rox, roy, roz, rox1, roy1,
(cont.) roz1, xp, yp, zp, xb, yb, zb
  integer kind
end type
type ionpos
  real x,y,z
  integer kind
end type
type (atom), allocatable:: atoms(:)
type (ionpos),allocatable :: ions(:)
integer n, ntype, nvac, nwall
real boxsize_x, boxsize_y, boxsize_z, rcut,a
integer ncellx,ncelly,ncellz
real cell_factorx, cell_factory, cell_factorz, bxr, byr, bzr
integer head_chain(0:1000), linked_list(40010),n_v_list(40000)
real atno(10), conc(10), vdd(3)
real,parameter:: avo = 6.022136736d23
end module commondefs
!! Declerations for MD
module vardecs
type lj
  real eps,sigma
end type
type fourier
  real a,b,c,d,e,f,g,h,ix,jx,kx,lx,mx,rmin,rmax,eps,sigma
end type
type poten
  integer index,pot
end type
type (poten):: pairtype(50)
type (lj):: ljpot(20)
type (fourier):: fourierpot(20)
integer totstep, np, enunit, scaleint, scalestep, nperf, g
real tote, totkin, hamilton, totw, volume, volume1, boxsize_x1, deltat,
(cont.) temp, tunit, tstp, tottime
real seed, tempdum, sumvel, tol
real nbox_x, nbox_y, nbox_z
real bolz
integer tstep, iconf, imovie, ivac, iforce, imsd, ielmigstep, msdstart,
(cont.) msdend
real q, ps, s
real eps, sigma, encorr
real pressure, pset, m
real mu, tau_t, tau_p, lambda, beta
real,dimension(10):: molwt
```

```
real,allocatable:: xini(:), yini(:), zini(:)
real,parameter:: pi = 3.14159265358979
end module vardecs
!! Declerations for EM force routine
Module Defs
save
type elements
  integer atno
  character(2) name
  real a0, a1, a2, rm, om, m, rc, alf, ec, c
  integer z
end type
type (elements) :: sp(10)
type (elements) :: el(56)
real rr, rrdir(3), omave, zave, mave, x, y, z, box_x, box_y, box_z,
(cont.) cutoff, xr, yr, zr
real kf, ef, vf, vd, vddir(3), rs, ksi
integer i1, i2, force_j, force_i
real,parameter::PI = 3.14159265358979
end module Defs
!! MD program
program md
use vardecs
use commondefs
real rp, r0, msd
real deltatsq, hdeltatsq, boxsize_x2, boxsize_y2, boxsize_z2
real elapsedtime
real pavr, sump
real totei, totkini
integer i, counter
real,external:: timef, read_control, open_output, fcc2, initvel,
(cont.) forceeval, nh1, nh2, elmig, new_nlist, findvac, vacf
character controlfile * 12
integer(2) status
  elapsedtime = timef()
  call getarg(1, controlfile, status)
  if (status .eq. -1) then
  write(*, *) "enter␣controlfile␣name␣:"
  read(*, *) controlfile
  if (controlfile == '') stop
  endif
  call read_control(controlfile)
  call open_output(controlfile,iconf,ivac,iforce,imsd,imovie)
  call fcc2(a,int(nbox_x),int(nbox_y),int(nbox_z))
  if (enunit .eq. 0) then
    bolz = 1.380658d-23
    tunit = 1.0d-15
    do i=1,ntype
      molwt(i) = molwt(i) / avo * 0.001
    enddo
  endif
  if(enunit .eq. 1) then
    bolz = 8.617317547d-5
    tunit = 0.09822732370761
  endif
!parameters in cut-off corrections
  eps = 1.507212225d-20
  sigma =  2.5748426d-10
  pset = 2.5d-3
  tau_p = 0.01
  tau_t = 1.0
  beta = 0.74474
  m = 1.0d-5
  volume1 = 0.0
  counter = 0
  !seed = 0.85
  np = n
  deltatsq = (deltat**2) * (tunit**2)
  hdeltatsq = deltatsq * 0.5
  tstp = deltat * tunit
  boxsize_x2 = boxsize_x / 2
  boxsize_y2 = boxsize_y / 2
```

```
   boxsize_z2 = boxsize_z / 2
   volume = boxsize_x * boxsize_y * boxsize_z
   bxr = boxsize_x / (volume**0.3333333333)
   byr = boxsize_y / (volume**0.3333333333)
   bzr = boxsize_z / (volume**0.3333333333)
!cut-off corrections
   encorr = (2.0 * pi * np / volume * 4.0 * eps ) * ((-1.0/9.0) * (sigma
(cont.)**12/rcut**9) - (-1.0/3.0) * (sigma**6/rcut**3))
!initialize property values
   msd = 0.0
   sumvel = 0.0
   pavr = 0.0
   !! Give initial velocities to atoms
   call initvel(temp)
   call forceeval(np, rcut, boxsize_x, boxsize_y, boxsize_z)
!!--::nose-hoover thermostat::--!!
   s = 0.0
   g = 3 * n
   ps = (sumvel - real(g) * bolz * temp) / q
! initial accelerations,second and third derivatives
   do i = 1, n
      atoms(i)%ax = atoms(i)%fx / molwt(atoms(i)%kind)
      atoms(i)%ay = atoms(i)%fy / molwt(atoms(i)%kind)
      atoms(i)%az = atoms(i)%fz / molwt(atoms(i)%kind)
      atoms(i)%bx = 0.0
      atoms(i)%bz = 0.0
      atoms(i)%by = 0.0
      atoms(i)%cx = 0.0
      atoms(i)%cy = 0.0
      atoms(i)%cz = 0.0
   end do
   write(5,1) "tottime", "poten", "kinetic", "toten", "temp", "pressure",
(cont.) "hamilton", "volume", "diffusion"
   1 format(10x, a7, 8a25)
   !!  MD Loop
!!!!!!!!!! ---- equilibration ------ !!!!!!!!!!
   do tstep = 1, scalestep
    !! Select the appropriate integration routine
    !! Un comment only one of the following, for the desired ensemble,
     !call predict(tstp) !! Uncomment this for Gear 5 value
     !call verlet1(tstp) !! Uncomment this for non-reversible Nose-Hoover
(cont.) Thermostat
     !call verl1(tstp)    !! Uncomment this for traditional velocity
(cont.) Verlet
     !call anders1()      !! Uncomment this for Andersen pressure control
     call nh1(tstp)       !! Uncomment this for reversible Nose-Hoover
(cont.) Thermostat
     !pbc
     do i = 1, n
       if (atoms(i)%x .lt. 0.0) then
         atoms(i)%x = atoms(i)%x + boxsize_x
       else if (atoms(i)%x .gt. boxsize_x) then
         atoms(i)%x = atoms(i)%x - boxsize_x
       end if
       !! Uncomment the following lines for PBC in y - direction
      !if (atoms(i)%y .lt. 0.0)then
        !atoms(i)%y = atoms(i)%y + boxsize_y
      !else if (atoms(i)%y .gt. boxsize_y) then
        !atoms(i)%y = atoms(i)%y - boxsize_y
      !end if
       if (atoms(i)%z .lt. 0.0) then
         atoms(i)%z = atoms(i)%z + boxsize_z
       else if (atoms(i)%z .gt. boxsize_z) then
         atoms(i)%z = atoms(i)%z - boxsize_z
       end if
     end do
     call forceeval(np,rcut,boxsize_x,boxsize_y,boxsize_z)
    !! Select the appropriate integration routine
    !! Uncomment only the corrector step of the previously selected
(cont.) integrator.
```

```
     call nh2(tstp)        !! Uncomment this for reversible Nose-Hoover
(cont.) Thermostat
     !call anders2()       !! Uncomment this for Andersen pressure control
     !call verlet2(tstp)   !! Uncomment this for non-reversible Nose-
(cont.) Hoover Thermostat
     !call correct(tstp)   !! Uncomment this for Gear 5 value
     !call berendsen()     !! Uncomment this for Berendsen pressure
(cont.) control
     !call verl2(tstp)     !! Uncomment this for traditional velocity
(cont.) Verlet
     tottime=counter*tstp
     !call velscale()      !! Uncomment this for temperature control by
(cont.) velocity scaling
     sumvel = 0.0
     do i = 1,n
       sumvel = sumvel + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz
(cont.) **2)*molwt(atoms(i)%kind)
     end do
     !! Uncomment the following in a pressure controlled simulation
     !boxsize_x = volume * bxr
     !boxsize_y = volume * byr
     !boxsize_z = volume * bzr
! calculate properties
     totkin = 0.5 * sumvel
     counter = counter + 1
     tempdum = (sumvel) / (3 * real(n) * bolz)
     pressure = ((n * bolz * tempdum) + (1 / 3) * totw) / volume
     sump = sump + pressure
     pavr = sump / counter
     totei = tote / ( n + nwall )
     totkini = totkin / ( n + nwall )
     hamilton = tote + totkin + ( ps**2 * q) / (2.0) + real(g) * tempdum
(cont.) * bolz * s
     write(*,2) tstep, tempdum, tote + totkin, hamilton
     2 format(i8,3es13.5)
!! update Linked list
     call new_nlist()
 !! Write the properties to a file
     if (enunit .eq. 1) then
       write(5,15) tottime, totei, totkini, totei + totkini + encorr,
(cont.) tempdum, pressure, hamilton, volume
       15 format(1x, 8es25.17)
     endif
     if (enunit .eq. 0)then
       write(5,16) tottime, totei, totkini, totei + totkini + encorr,
(cont.) tempdum, pressure, hamilton, volume
       16 format(1x, 8es25.17)
     endif
   !! write the positions, velocities, accelerations and vacancy
(cont.) positions at intervals
     if (ivac .gt. 0) then
       if (mod(tstep, ivac) .eq. 0) then
         write(8, *) nvac
         write(8, *) "vacancy␣positions:␣", "at␣step=␣", tstep,"␣time=␣",
(cont.) tottime,"␣fs"
         do i = n + nwall + 1, n + nwall + nvac
           write(8, 14) atoms(i)%x, atoms(i)%y, atoms(i)%z
           14 format(1x, 3f20.10)
         enddo
       endif
     endif
   !! Write the force on the atoms at intervals
     if (iforce .gt. 0) then
       if (mod(tstep, iforce) .eq. 0) then
         write(7, *) "forces␣", "at␣step=␣", tstep,"␣time=␣",tottime,"␣fs
(cont.) "
         do i = 1,n + nwall + nvac
           write(7, 13) atoms(i)%fx, atoms(i)%fy, atoms(i)%fz, atoms(i)%
(cont.) elfx, atoms(i)%elfy, atoms(i)%elfz
           13 format(1x,6es15.7)
```

```
        enddo
      endif
    endif
 !! write the trajectory of atoms to .xyz file
    if (imovie .gt. 0) then
      if (mod(tstep, imovie) .eq. 0) then
        write(15, *) n + nwall
        write(15, *) 'md step ', tstep
        do i = 1, n + nwall
          write(15, '(i3,3f10.4)') atoms(i)%kind, atoms(i)%x,atoms(i)%y,
(cont.) atoms(i)%z
        enddo
      endif
    endif
  end do
!!!!!! ---- production ------ !!!!!!!!!!
!store initial positions for MSD calculations
  do i = 1, n
    atoms(i)%x0 = atoms(i)%x
    atoms(i)%y0 = atoms(i)%y
    atoms(i)%z0 = atoms(i)%z
  end do
  do tstep = scalestep + 1, totstep
!! store first positions for msd
    do i = 1, n
      atoms(i)%xb = atoms(i)%x
      atoms(i)%yb = atoms(i)%y
      atoms(i)%zb = atoms(i)%z
    enddo
    call nh1(tstp)        !! Uncomment this forreversible Nose-Hoover
(cont.) Thermostat
    !call predict(tstp)   !! Uncomment this for Gear 5 value
    !call verlet1(tstp)   !! Uncomment this for non-reversible Nose-
(cont.) Hoover Thermostat
    !call verl1(tstp)     !! Uncomment this for traditional velocity
(cont.) Verlet
!! ----- pbc -----
    do i = 1, n
      if (atoms(i)%x .lt. 0.0) then
        atoms(i)%x = atoms(i)%x + boxsize_x
      else if (atoms(i)%x .gt. boxsize_x) then
        atoms(i)%x = atoms(i)%x - boxsize_x
      end if
      !! Uncomment the following lines for PBC in y- direction
     !if (atoms(i)%y .lt. 0.0)then
       !atoms(i)%y = atoms(i)%y + boxsize_y
     !else if (atoms(i)%y .gt. boxsize_y) then
       !atoms(i)%y = atoms(i)%y - boxsize_y
     !end if
      if (atoms(i)%z .lt. 0.0) then
        atoms(i)%z = atoms(i)%z + boxsize_z
      else if (atoms(i)%z .gt. boxsize_z) then
        atoms(i)%z = atoms(i)%z - boxsize_z
      end if
    end do
    !! Evaluate Forces and calculate energy
    call forceeval(np,rcut,boxsize_x,boxsize_y,boxsize_z)
    if (mod(tstep,ielmigstep)==0) then
     !! Find vaacancies
      call findvac()
     !! add electromigration force !!!!!
      call elmig()
    end if
     !! EM force to MD force
    do i = 1, n
      atoms(i)%fx = atoms(i)%fx + atoms(i)%elfx
      atoms(i)%fy = atoms(i)%fy + atoms(i)%elfy
      atoms(i)%fz = atoms(i)%fz + atoms(i)%elfz
    end do
   !! Select the appropriate integration routine
   !! Uncomment only the corrector step of the previously selected
```

```
(cont.) integrator.
     call nh2(tstp)        !! Uncomment this for non-reversible Nose-
(cont.) Hoover Thermostat
     !call verlet2(tstp)   !! Uncomment this for non-reversible Nose-
(cont.) Hoover Thermostat
     !call correct(tstp)   !! Uncomment this for Gear 5 value
     !call verl2(tstp)     !! Uncomment this for traditional velocity
(cont.) Verlet
     tottime = counter * tstp
     if (imsd .ne. 0) then
       if (mod(tstep,imsd) .eq. 0) then
         msd = 0.0
     !! rewind pbc for msd calculation
         do i = 1, n
           atoms(i)%xp = atoms(i)%x + dnint((atoms(i)%xb - atoms(i)%x) /
(cont.) boxsize_x) * boxsize_x
           atoms(i)%yp = atoms(i)%y + dnint((atoms(i)%yb - atoms(i)%y) /
(cont.) boxsize_y) * boxsize_y
           atoms(i)%zp = atoms(i)%z + dnint((atoms(i)%zb - atoms(i)%z) /
(cont.) boxsize_z) * boxsize_z
         enddo
     !! MSD calculation
         do i=msdstart,msdend
           rp=dsqrt(atoms(i)%xp**2 + atoms(i)%yp**2 + atoms(i)%zp**2)
           r0=dsqrt(atoms(i)%x0**2 + atoms(i)%y0**2 + atoms(i)%z0**2)
           msd=msd + (rp-r0)**2
         enddo
         msd = msd / real(msdend-msdstart+1)
         write(4,'(f20.13,1x,es25.17)') real((tstep - (scalestep+imsd)))*
(cont.) tstp,msd
       endif
     endif
     sumvel = 0.0
     do i = 1, n
       sumvel = sumvel + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz
(cont.) **2) * molwt(atoms(i)%kind)
     end do
! calculate properties
     totkin = 0.5 * sumvel
     counter = counter + 1
     tempdum = (sumvel) / (3 * real(n) * bolz)
     pressure = ((n * bolz * tempdum) + (1/3) * totw) / volume
     sump = sump + pressure
     pavr = sump / counter
     totei = tote / ( n + nwall )
     totkini = totkin / ( n + nwall )
     hamilton = tote + totkin + ( ps**2 * q) / (2.0) + real(g) * tempdum
(cont.) * bolz * s
     write(*, 3) tstep, tempdum, tote, hamilton
     3 format(i8, 3es13.5)
   !! Update neighbor list
     call new_nlist()
     if (enunit .eq. 1) then
       write(5, 18) tottime, totei, totkini, totei + totkini + encorr,
(cont.) tempdum, pressure, hamilton, volume, msd
      18 format(1x, 9es25.17)
     endif
     if (enunit .eq. 0)then
       write(5, 19) tottime, totei, totkini, totei + totkini + encorr,
(cont.) tempdum, pressure, hamilton, volume, msd
       19 format(1x,9es25.17)
     endif
   !! write the positions, velocities, accelerations and vacancy
(cont.) positions at intervals
     if (iconf .gt. 0) then
       if (mod(tstep, iconf) .eq. 0) then
         write(9, *) tstep
         do i = 1, n
           write(9, 21) atoms(i)%x, atoms(i)%y, atoms(i)%z, atoms(i)%vx,
```

```
(cont.) atoms(i)%vy, atoms(i)%vz
           21 format(1x, 6es15.7)
         end do
       endif
     endif
     if (iforce .gt. 0) then
       if (mod(tstep, iforce) .eq. 0) then
         write(7, *) "forces ", "at step= ", tstep, " time= ", tottime, "
(cont.) fs"
         do i = 1, n + nvac + nwall
           write(7, 24) atoms(i)%fx, atoms(i)%fy, atoms(i)%fz, atoms(i)%
(cont.) elfx, atoms(i)%elfy, atoms(i)%elfz
           24 format(1x,6es15.7)
         enddo
       endif
     endif
     if (ivac .gt. 0) then
       if (mod(tstep, ivac) .eq. 0) then
         write(8, *) nvac
         write(8, *) "vacancy positions: ", "at step= ", tstep," time= ",
(cont.) tottime," fs"
         do i = n + nwall + 1, n + nvac + nwall
           write(8, 22) atoms(i)%x, atoms(i)%y, atoms(i)%z
           22 format(1x, 3f20.10)
         enddo
       endif
     endif
  !! write the trajectory of atoms to .xyz file
     if (imovie .gt.0 ) then
       if (mod(tstep, imovie) .eq. 0) then
         write(15, *) n + nwall
         write(15, *) 'md step ', tstep
         do i = 1, n + nwall
           write(15, '(i3,3f10.4)') atoms(i)%kind, atoms(i)%x, atoms(i)%y
(cont.), atoms(i)%z
         enddo
       endif
     endif
   enddo
   if (iconf .gt. 0) close(9)
   if (imsd .gt. 0)  close(4)
   if (iforce .gt. 0)  close(7)
   if (ivac .gt. 0)  close(8)
   if (imovie .gt. 0)  close(15)
   !! Calculate velocity autocorrelation function
   call vacf(controlfile, totstep, scalestep, tstp, n, iconf)
   elapsedtime = timef()
   write(*, *) "total elapsed time is: ", elapsedtime
   write(5, *) "total elapsed time is:", elapsedtime, " secs."
   close(5)
end program md
!! This routine opens output files
subroutine open_output(name1, ic, iv, ifrc, ims, imov)
integer ic, iv, ifrc, ims, imov
character name1*(*)
character(len = len_trim(name1)) name2
   name2 = trim(name1)
   open(unit = 5, file = name2//".out")
   if (ic .gt. 0) open(unit = 9, file = name2//".con")
   if (iv .gt. 0) open(unit = 8, file = name2//".vac")
   if (ifrc .gt. 0) open(unit = 7, file = name2//".elf")
   if (ims .gt. 0) open(unit = 4, file = name2//".msd")
   if(imov .gt. 0) open(unit = 15, file = name2//".xyz")
end subroutine open_output
!! Read input data
subroutine read_control(name1)
use vardecs
use commondefs
```

```
integer i
character name1*(*), name*22
character(len = len_trim(name1)) name2
integer npot, pottype, at1, at2, bit
  name2 = trim(name1)
  open(unit = 10, file = name2)
  open(unit = 11, file = 'fcc.txt')
  read(10, *) n, nvac, nwall
  read(10, *) ntype
  read(10, *) totstep, scalestep, scaleint
  read(10, *) deltat
  read(10, *) tol
  read(10, *) msdstart, msdend
  read(10, *) temp
  read(10, *) q
  read(10, *) iconf, ivac, iforce, imsd, imovie, ielmigstep
  read(10, *) enunit
  read(10, *) a, boxsize_x, boxsize_y, boxsize_z, rcut
  read(10, *) (molwt(i), i = 1, ntype)
  do i = 1, ntype
    read(10, *) atno(i), conc(i)
  enddo
  read(10, *) vdd(1), vdd(2), vdd(3)
  allocate(atoms(n + nvac + nwall + 150))
  allocate(ions(n + nvac + nwall + 150))
  do i = 1, 50
    pairtype(i)%pot = 0
  enddo
  read(10, *) npot
  do i = 1,npot
    read(10, *) pottype
    select case (pottype)
      case (1) !lj
        read (10, *) at1, at2, ljpot(i)%eps, ljpot(i)%sigma
        bit = 2**at1 + 2**at2
        pairtype(bit)%index = i
        pairtype(bit)%pot = 1
      case(2) ! fourier potential
        read(10, *) at1, at2, fourierpot(i)%a, fourierpot(i)%b,
(cont.) fourierpot(i)%c, fourierpot(i)%d, fourierpot(i)%e, fourierpot(i)%f,
(cont.) fourierpot(i)%g, fourierpot(i)%h, fourierpot(i)%ix, fourierpot(i)%jx
(cont.), fourierpot(i)%kx, fourierpot(i)%lx, fourierpot(i)%mx, fourierpot(i
(cont.))%rmin, fourierpot(i)%rmax, fourierpot(i)%eps, fourierpot(i)%sigma
        bit=2**at1 + 2**at2
        pairtype(bit)%index = i
        pairtype(bit)%pot = 2
      end select
  enddo
  nbox_x = int(boxsize_x / a)
  nbox_y = int(boxsize_y / a)
  nbox_z = int(boxsize_z / a)
  nperf = nbox_x * nbox_y * nbox_z * 4
! read atomic coordinates
  do i = 1,n + nvac + nwall
    read(10,10) atoms(i)%kind, atoms(i)%x, atoms(i)%y, atoms(i)%z
    10 format(i2, 1x, f20.17, 4x, f20.17, 4x, f20.17)
  end do
  allocate(xini(nperf), yini(nperf), zini(nperf))
  close(11)
  ncellx = int(boxsize_x / rcut / 1.01)
  ncelly = int(boxsize_y /rcut / 1.01)
  ncellz = int(boxsize_z / rcut / 1.01)
  cell_factorx = ncellx / boxsize_x
  cell_factory = ncelly / boxsize_y
  cell_factorz = ncellz / boxsize_z
  call new_nlist()
```

```
   call neighbour_cells()
end subroutine read_control
!! This routine gives initial velocities to atoms
subroutine initvel (temper)
use commondefs
use vardecs
real rtemp, sumx, sumy, sumz
real gauss, dummy,temper,scale,sumvel2
real aheat2
integer i
  dummy = 0.5
  rtemp = dsqrt (temper)
  aheat2 = real(n) * 3.0 * bolz * temper
  do i = 1, n
    atoms(i)%vx = rtemp * gauss (dummy)
    atoms(i)%vy = rtemp * gauss (dummy)
    atoms(i)%vz = rtemp * gauss (dummy)
  end do
  sumx = 0.0
  sumy = 0.0
  sumz = 0.0
  do i = 1, n
    sumx = sumx + atoms(i)%vx
    sumy = sumy + atoms(i)%vy
    sumz = sumz + atoms(i)%vz
  end do
  sumx = sumx / real (n)
  sumy = sumy / real (n)
  sumz = sumz / real (n)
  do i = 1, n
    atoms(i)%vx = atoms(i)%vx - sumx
    atoms(i)%vy = atoms(i)%vy - sumy
    atoms(i)%vz = atoms(i)%vz - sumz
  end do
  sumvel2 = 0.0
  do i = 1,n
    sumvel2 = sumvel2 + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz
(cont.) **2)*molwt(atoms(i)%kind)
  end do
  scale = dsqrt(aheat2 / sumvel2)
  sumvel = 0.0
  do i = 1, np
    atoms(i)%vx = atoms(i)%vx * scale
    atoms(i)%vy = atoms(i)%vy * scale
    atoms(i)%vz = atoms(i)%vz * scale
    sumvel=sumvel + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz**2) *
(cont.) molwt(atoms(i)%kind)
  end do
return
end subroutine initvel
!! This function gives a random number in Gaussian distribution
real function gauss (dummy)
real a1, a3, a5, a7, a9
parameter (a1 = 3.949846138, a3 = 0.252408784)
parameter (a5 = 0.076542912, a7 = 0.008355968)
parameter (a9 = 0.029899776)
real sum, r, r2
real ranf, dummy
integer i
  sum = 0.0
  do i = 1, 12
    sum = sum + ranf(dummy)
  enddo
  r  = (sum - 6.0) / 4.0
  r2 = r * r
  gauss = ((((a9 * r2 + a7) * r2 + a5) * r2 + a3) * r2 +a1)*r
return
end function gauss
!! This function generates  a random number
```

```
real function ranf (dummy)
integer l, c, m
parameter (l = 1029, c = 221591, m = 1048576)
integer seed
real dummy
save seed
data seed / 0 /
  seed = mod (seed * l + c, m)
  ranf = real (seed) / m
return
end function ranf
!! Reversible Nose-Hoover thermostat with velocity Verlet predictor step
subroutine nh1(dt)
use vardecs
use commondefs
real dt, dtsq2, dt2, sumvel2
integer i
  dt2   = dt / 2.0
  dtsq2 = dt * dt2
  sumvel2 = 0.0
  do i = 1, n
    sumvel2 = sumvel2 + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz
(cont.) **2) * molwt(atoms(i)%kind)
  end do
  ps = ps + (sumvel2 - real(g) * bolz * temp) * dt2  / q
  s = s + ps * dt
  do i = 1,n
    atoms(i)%vx = atoms(i)%vx * dexp(-ps * dt2)+ (atoms(i)%fx / molwt(
(cont.) atoms(i)%kind)) * dt2
    atoms(i)%vy = atoms(i)%vy * dexp(-ps * dt2)+ (atoms(i)%fy / molwt(
(cont.) atoms(i)%kind)) * dt2
    atoms(i)%vz = atoms(i)%vz * dexp(-ps * dt2)+ (atoms(i)%fz / molwt(
(cont.) atoms(i)%kind)) * dt2
    atoms(i)%x = atoms(i)%x + dt * atoms(i)%vx
    atoms(i)%y = atoms(i)%y + dt * atoms(i)%vy
    atoms(i)%z = atoms(i)%z + dt * atoms(i)%vz
  end do
end subroutine nh1
!! Reversible Nose-Hoover thermostat with velocity Verlet predictor step
subroutine nh2(dt)
use vardecs
use commondefs
real  dt, dt2, sumvel2
integer i
  dt2 = dt / 2.0
  sumvel2 = 0.0
  totkin = 0.0
  do i = 1, n
    atoms(i)%vx = (atoms(i)%vx + (atoms(i)%fx / molwt(atoms(i)%kind)) *
(cont.) dt2) * dexp(-ps * dt2)
    atoms(i)%vy = (atoms(i)%vy + (atoms(i)%fy / molwt(atoms(i)%kind)) *
(cont.) dt2) * dexp(-ps * dt2)
    atoms(i)%vz = (atoms(i)%vz + (atoms(i)%fz / molwt(atoms(i)%kind)) *
(cont.) dt2) * dexp(-ps * dt2)
    sumvel2 = sumvel2 + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz
(cont.) **2) * molwt(atoms(i)%kind)
  end do
  ps = ps + (sumvel2-real(g)* bolz * temp) * dt2 / q
end subroutine nh2
!! This routine calculates the forces from of the interatomic potential
(cont.) defined on atoms in MD
subroutine forceeval(npart, rcut2, box_x, box_y, box_z)
use vardecs
use commondefs
real rcut2, box_x, box_y, box_z, rcutsq2, box_x2, box_y2, box_z2, xr, yr
(cont.), zr, xi , yi, zi
real, external:: forcefunc, potfunc, lennardforce, lennardenergy
integer,external :: find_cell
real rxij, ryij, rzij, rijsq, rijsqrt
```

```
real fij, rpl, fxij, fyij, fzij
integer npart, bit, i, j,ic, inn, in
  tote = 0.0
  totw = 0.0
  rcutsq2 = rcut2**2
  box_x2 = box_x / 2
  box_y2 = box_y / 2
  box_z2 = box_z / 2
  do i = 1, npart + nwall
    atoms(i)%fx = 0.0
    atoms(i)%fy = 0.0
    atoms(i)%fz = 0.0
  end do
  do i = 1, npart + nwall - 1
    xi = atoms(i)%x
    yi = atoms(i)%y
    zi = atoms(i)%z
    ic = find_cell(atoms(i)%x, atoms(i)%y, atoms(i)%z)
    do inn = int(ic * 27 + 1), int(ic * 27 + 27)
      in = n_v_list(inn)
      j = head_chain(in)
      do while (j /= 0)
        if ((j /= i) .and. (j >= i + 1)) then
          bit = 2**atoms(i)%kind + 2**atoms(j)%kind
          if (pairtype(bit)%pot == 0) then
            j = linked_list(j)
            cycle
          endif
          rxij = xi - atoms(j)%x
          ryij = yi - atoms(j)%y
          rzij = zi - atoms(j)%z
    ! Minimum image convention
          xr = abs(rxij)
          yr = abs(ryij)
          zr = abs(rzij)
          if (xr .gt. box_x2) rxij = (xr - box_x) * sign(1.0, rxij)
          if (yr .gt. box_y2) ryij = (yr - box_y) * sign(1.0, ryij)
          if (zr .gt. box_z2) rzij = (zr - box_z) * sign(1.0, rzij)
          rijsq = (rxij * rxij) + (ryij * ryij) + (rzij * rzij)
          rijsqrt = dsqrt(rijsq)
          if(rijsq .lt. rcutsq2)then
            select case(pairtype(bit)%pot)
              case (1)
                fij = lennardforce(rijsqrt,bit)
                tote = tote + lennardenergy(rijsqrt,bit)
              case (2)
                fij = forcefunc(rijsqrt,bit)
                tote = tote + potfunc(rijsqrt,bit)
            end select
            rpl = fij * rijsqrt
            fxij = fij * rxij / rijsqrt
            fyij = fij * ryij / rijsqrt
            fzij = fij * rzij / rijsqrt
            atoms(i)%fx = atoms(i)%fx + fxij
            atoms(j)%fx = atoms(j)%fx - fxij
            atoms(i)%fy = atoms(i)%fy + fyij
            atoms(j)%fy = atoms(j)%fy - fyij
            atoms(i)%fz = atoms(i)%fz + fzij
            atoms(j)%fz = atoms(j)%fz - fzij
            totw = totw + rpl
          end if
        endif
        j = linked_list(j)
      enddo
    end do
  end do
return
```

```
end subroutine forceeval
!! This function calculates potential energy by Fourier series
function potfunc(rij, bit)
use vardecs
real potfunc
real rij, rx
integer bit
  rx = pi * (rij - fourierpot(pairtype(bit)%index)%rmin) / (fourierpot(
(cont.) pairtype(bit)%index)%rmax - fourierpot(pairtype(bit)%index)%rmin)
  potfunc = fourierpot(pairtype(bit)%index)%a + fourierpot(pairtype(bit)
(cont.)%index)%b * cos(rx) + fourierpot(pairtype(bit)%index)%c * sin(rx) +
(cont.) fourierpot(pairtype(bit)%index)%d * cos(2.0 * rx) + fourierpot(
(cont.) pairtype(bit)%index)%e * sin(2.0 * rx) + fourierpot(pairtype(bit)%
(cont.) index)%f * cos(3.0 * rx) + fourierpot(pairtype(bit)%index)%g * sin
(cont.) (3.0 * rx) + fourierpot(pairtype(bit)%index)%h * cos(4.0 * rx) +
(cont.) fourierpot(pairtype(bit)%index)%ix * sin(4.0 * rx) + fourierpot(
(cont.) pairtype(bit)%index)%jx * cos(5.0 * rx) + fourierpot(pairtype(bit)%
(cont.) index)%kx * sin(5.0 * rx) + fourierpot(pairtype(bit)%index)%lx * cos
(cont.) (6.0 * rx) + fourierpot(pairtype(bit)%index)%mx * sin(6.0 *rx)
return
end function potfunc
!! This function calculates the force by Fourier series
function forcefunc(rij2, bit)
use vardecs
real forcefunc
real rij2, y, ry
integer bit
ry = pi * (rij2 - fourierpot(pairtype(bit)%index)%rmin) / (fourierpot(
(cont.) pairtype(bit)%index)%rmax - fourierpot(pairtype(bit)%index)%rmin)
y = pi / (fourierpot(pairtype(bit)%index)%rmax - fourierpot(pairtype(bit
(cont.) )%index)%rmin)
forcefunc =  fourierpot(pairtype(bit)%index)%b * sin(ry) * y -
(cont.) fourierpot(pairtype(bit)%index)%c * cos(ry) * y + 2.0 * fourierpot(
(cont.) pairtype(bit)%index)%d * sin(2.0 * ry) * y - 2.0 * fourierpot(
(cont.) pairtype(bit)%index)%e * cos(2.0 * ry) * y + 3.0 * fourierpot(
(cont.) pairtype(bit)%index)%f * sin(3.0 * ry) * y - 3.0 * fourierpot(
(cont.) pairtype(bit)%index)%g * cos(3.0 * ry) * y + 4.0 * fourierpot(
(cont.) pairtype(bit)%index)%h * sin(4.0 * ry) * y - 4.0 * fourierpot(
(cont.) pairtype(bit)%index)%ix * cos(4.0 * ry) * y + 5.0 * fourierpot(
(cont.) pairtype(bit)%index)%jx * sin(5.0 * ry) * y - 5.0 * fourierpot(
(cont.) pairtype(bit)%index)%kx * cos(5.0 * ry) * y + 6.0 * fourierpot(
(cont.) pairtype(bit)%index)%lx * sin(6.0 * ry) * y - 6.0 * fourierpot(
(cont.) pairtype(bit)%index)%mx * cos(6.0 * ry) * y
return
end function forcefunc
!! Calculate forces if Lennard-Jones potential is used
function lennardforce(rij3, bit)
use vardecs
integer bit
real lennardforce, rij3
lennardforce = 24 * ljpot(pairtype(bit)%index)%eps * (2 * (ljpot(
(cont.) pairtype(bit)%index)%sigma**12 / rij3**13) - (ljpot(pairtype(bit)%
(cont.) index)%sigma**6 / rij3**7))
return
end function lennardforce
!! This function calculates the potential energy if Lennard-Joned
(cont.) potential is used
function lennardenergy(rij,bit)
use vardecs
integer bit
real lennardenergy,rij
lennardenergy = 4 * ljpot(pairtype(bit)%index)%eps *((ljpot(pairtype(bit
(cont.) )%index)%sigma / rij)**12 - (ljpot(pairtype(bit)%index)%sigma / rij)
(cont.) **6)
return
end function lennardenergy
!! This routine finds the cell of atom in linked list
integer function find_cell(ionx, iony, ionz)
```

```
use commondefs
real ionx, iony, ionz
integer ix, iy, iz
  ix = int(ionx * cell_factorx)
  iy = int(iony * cell_factory)
  iz = int(ionz * cell_factorz)
  find_cell = ix + iy * ncellx + iz * ncellx * ncelly
end function find_cell
!! Linked list
subroutine neighbour_cells()
use commondefs
integer ix, iy, iz, ic, in, itel, icx, icy, icz, iccx, iccy, iccz
  do ic = 0, ncellx * ncelly * ncellz - 1
    itel = 0
    icz = ic / (ncellx * ncelly)
    icy = (ic - icz * ncellx * ncelly) / ncellx
    icx = (ic - icy * ncellx - icz * ncellx * ncelly)
    do iz = -1, 1
      iccz = icz + iz
      if (iccz < 0) then
        iccz=iccz+ncellz
      else if (iccz >= ncellz) then
        iccz = iccz - ncellz
      endif
      do iy = -1, 1
        iccy = icy + iy
        if (iccy < 0) then
          iccy = iccy + ncelly
        else if (iccy >= ncelly) then
          iccy = iccy - ncelly
        endif
        do ix = -1, 1
          iccx=icx+ix
          if (iccx < 0) then
            iccx = iccx + ncellx
          else if (iccx >= ncellx) then
            iccx = iccx - ncellx
          endif
          in = (iccx + iccy * ncellx + iccz * ncellx * ncelly)
          itel = itel + 1
          n_v_list(int(ic * 27 + itel)) = in
        enddo
      enddo
    enddo
  enddo
end subroutine neighbour_cells
!! This routine construct linked list
subroutine new_nlist()
use commondefs
integer i, ic
integer, external :: find_cell
  do ic = 0, ncellx * ncelly * ncellz - 1
    head_chain(ic) = 0
  enddo
  do i = 1, n + nwall
    ic = find_cell(atoms(i)%x, atoms(i)%y, atoms(i)%z)
    linked_list(i) = head_chain(ic)
    head_chain(ic) = i
  enddo
end subroutine new_nlist
!!! This routine is the initialization part of the electromigration
(cont.) program
subroutine init()
use defs
use commondefs
integer i,j
  vd = dsqrt(vdd(1)**2 + vdd(2)**2 + vdd(3)**2)
  vddir(1) = vdd(1) / vd
  vddir(2) = vdd(2) / vd
  vddir(3) = vdd(3) / vd
  do i = 1, ntype
```

```
      do j = 1, 56
        if (atno(i) .eq. el(j)%atno) sp(i) = el(j)
      enddo
    enddo
    box_x = boxsize_x / 0.529177
    box_y = boxsize_y / 0.529177
    box_z = boxsize_z / 0.529177
    cutoff = rcut / 0.529177
    do i = 1, n + nvac + nwall
     ions(i)%x = atoms(i)%x / 0.529177249
     ions(i)%y = atoms(i)%y / 0.529177249
     ions(i)%z = atoms(i)%z / 0.529177249
     ions(i)%kind = atoms(i)%kind
    enddo
    close(6)
    omave = 0.0
    zave = 0.0
    mave = 0.0
    do i= 1, ntype
      omave = omave + conc(i) * sp(i)%om
      zave = zave + conc(i) * sp(i)%z
      mave = mave + conc(i) * sp(i)%m
    enddo
    mave = 1.0
    kf = (3 * pi * pi * zave / omave)**0.33333333333333
    vf = kf / mave
    ef = kf * kf / 2 / mave
    rs = mave * (3.0 * omave / 4.0 / pi / zave)**0.33333333333333
    ksi = 2.0 / (1.0 + 0.02544716775 * rs)
    do i= 1, n + nwall
      atoms(i)%elfx = 0.0
      atoms(i)%elfy = 0.0
      atoms(i)%elfz = 0.0
    enddo
end
!! This function calculates the cosine of the angle between vd and Rj-Rm
real function coss(rx, ry, rz)
use defs
real rx, ry, rz
  coss = rx / rr * vddir(1) + ry / rr * vddir(2) + rz / rr * vddir(3)
end function coss
!! This function calculates the dot product of vd and Rj
real function dotproduct(ii, var)
use defs
use commondefs
real var
integer ii
  if (ii .eq. 1) dotproduct = vdd(1) * var + vdd(2) * y + vdd(3) * z
  if (ii .eq. 2) dotproduct = vdd(1) * x + vdd(2) * var + vdd(3) * z
  if (ii .eq. 3) dotproduct = vdd(1) * x + vdd(2) * y + vdd(3) * var
end function dotproduct
!! This function calculates the spherical Bessel function
real function bessel(q)
use defs
real xx, q
  xx = q * rr
  bessel = sin(xx) / xx / xx - cos(xx) / xx
end function bessel
!! This function is used for the integration in the force calculation
real function integrant1(q)
use defs
use commondefs
real wi, wj, q
real, external :: w,bessel
  wj = w(sp(ions(force_j)%kind),q)
  if(ions(force_i)%kind .lt. 0)then
    wi = w(sp(1),q)
  else
    wi = w(sp(ions(force_i)%kind),q)
  endif
```

```
  integrant1 = wj * wi * q * q * bessel(q)
end function integrant1
real function integrant2(q)
use defs
use commondefs
real wj, wi, q
real, external :: w
  wj = w(sp(ions(force_j)%kind),q)
  wi = w(sp(1),q)
  integrant2 = wj * (wj - wi) * q * q * q
end function integrant2
real function integrant3(q)
use defs
use commondefs
real wi, wj, q
real, external :: w, bessel
  wi = w(sp(1),q)
  wj = w(sp(ions(force_j)%kind),q)
  integrant3 = wi * wj * q * q * bessel(q)
end function integrant3
!! This is the integration routine with Gauss quadrature method
real function integrate(integrant)
use defs
real a, b, epsabs, epsrel, result, abserr
integer key, neval, ier
real,external:: integrant
external qag
  a = 0.00001
  b = 2.0 * kf - 0.00001
  epsabs = 0.0
  epsrel = 0.001
  key = 3
  call qag(integrant, a, b, epsabs, epsrel, key, result, abserr, neval,
(cont.) ier)
  integrate = result
end function integrate
!! This function calculates the potential used in EM force calculation
real function uu_s(ii,var)
use defs
use commondefs
real, external :: coss, dotproduct, integrant1, integrant2, integrant3,
(cont.) integrate
real t1, term1, term2, term3, uu, var, rx, ry, rz
integer ii
  t1 = -omave**2 * mave * kf / 4.0 / pi**3
  uu = 0.0
  do force_i = 1, n + nvac + nwall
    if (ions(force_i)%kind .ne. 1) then
      rx = ions(force_i)%x - x
      ry = ions(force_i)%y - y
      rz = ions(force_i)%z - z
      if (ii .eq. 1) rx = ions(force_i)%x - var
      if (ii .eq. 2) ry = ions(force_i)%y - var
      if (ii .eq. 3) rz = ions(force_i)%z - var
      xr = dabs(rx)
      yr = dabs(ry)
      zr = dabs(rz)
      if (xr .gt. box_x / 2) rx = (xr - box_x) * dsign(1.0, rx)
          if (yr .gt. box_y / 2) ry = (yr - box_y) * dsign(1.0, ry)
      if (zr .gt. box_z / 2) rz = (zr - box_z) * dsign(1.0, rz)
      rr = dsqrt(rx * rx + ry * ry + rz * rz)
      if (rr <= cutoff) then
        if (force_i .ne. force_j) then
          term1 = t1 * vd / vf * coss(-rx, -ry, -rz) * integrate(
(cont.) integrant1)
          if (ions(force_i)%kind .lt. 0) then
            uu = uu - term1
```

```
          else
            uu = uu + term1
          endif
          if (ions(force_i)%kind .gt. 1) then
            term3 = t1 * vd / vf * coss(-rx, -ry, -rz) * integrate(
(cont.)integrant3)
            uu = uu - term3
          endif
        else
          term2 = t1 / 3.0 * dotproduct(ii,var) / vf * integrate(
(cont.)integrant2)
          uu = uu + term2
        endif
        endif
      endif
    enddo
    uu_s = uu
end function uu_s
!! This is the main EM force routine
subroutine elmig()
use defs
use commondefs
real error
real, external :: uu_s, dfridr, init
include 'data.h'
!! All units in EM force is in a.u.
  call init()
  do force_j = 1, n
    x = ions(force_j)%x
    y = ions(force_j)%y
    z = ions(force_j)%z
    atoms(force_j)%elfx = -dfridr(1, x, 0.2, error) * 51.42210754   !
(cont.)convert to eV/A
    atoms(force_j)%elfy = -dfridr(2, y, 0.2, error) * 51.42210754
    atoms(force_j)%elfz = -dfridr(3, z, 0.2, error) * 51.42210754
  enddo
end subroutine elmig
!! This function calculates Heine-Abarenkov-Aniamalu psp N(q)
real function bofq(ele, q)
use defs
real q, qrm, qrc, t1, t2, t3, t4, t5, t6, t7
type (elements) :: ele
  qrm = q * ele%rm
  qrc = q * ele%rc
  t1 = pi / omave / q / q
  t2 = 8.0 * t1 * ele%c / q
  t3 = sin(qrm) - qrm * cos(qrm)
  t4 = 8.0 * t1 * ele%z * cos(qrm)
  t5 = 4.0 * t1 * ele%ec / q
  t6 = 24.0 * t1 * ele%z * ele%alf / qrc / qrc / qrc
  t7 = sin(qrc) - qrc * cos(qrc)
  bofq =-t2 * t3 - t4 + (t5 - t6) * t7
end function bofq
!! This function calculates Heine-Abarenkov-Aniamalu psp N(q)
real function nofq(ele, q)
use defs
real q, t1, t2, t3, t4, xx, yy, cost, costp
real j0x, j1x, j2x, j3x, j0y, j1y, j2y, j3y
type (elements) :: ele
  t1 = pi * ele%rm * ele%rm * ele%rm / omave
  xx = kf * ele%rm
  j0x = sin(xx) / xx
  j1x = sin(xx) / xx / xx - cos(xx) / xx
  j2x = 3.0 / xx * j1x - j0x
  j3x = 5.0 / xx * j2x - j1x
  if ((q - 2 * kf) .le. 0.0) then
    cost = (1.0 - q * q / 2.0 / kf / kf)
    t2 = -4.00 * t1 * (ele%a0 - ele%c) * (j0x * j0x - j1x * cos(xx)/xx)
    t3 = -12.0 * t1 * (ele%a1 - ele%c) * (j1x * j1x - j0x * j2x) * cost
    t4 = -20.0 * t1 * (ele%a2 - ele%c) * (j2x * j2x - j1x * j3x) * (3.0
```

```
(cont.)* cost * cost - 1) / 2.0
  else
    yy = (q - kf) * ele%rm
    j0y = sin(yy) / yy
    j1y = sin(yy) / yy / yy - cos(yy) / yy
    j2y = 3.0 / yy * j1y - j0y
    j3y = 5.0 / yy * j2y - j1y
    costp = (xx * xx + yy * yy - (q * ele%rm)**2) / 2 / xx / yy
    t2 = -8.00 * t1 * (ele%a0 - ele%c) / (xx * xx - yy * yy) * (xx * j1x
(cont.) * j0y - yy * j1y * j0x)
    t3= -24.0 * t1 * (ele%a1 - ele%c) / (xx * xx - yy * yy) * (xx * j2x
(cont.)* j1y - yy * j2y * j1x) * costp
    t4 = -40.0 * t1 * (ele%a2 - ele%c) / ( xx * xx - yy * yy) * (xx *
(cont.)j3x * j2y - yy * j3y * j2x) * (3.0 * costp * costp - 1.0) / 2.0
  end if
  nofq = t2 + t3 + t4
return
end function nofq
!! This calculation calculates the unscreened  HAA form factor
real function w0(ele, q)
use defs
real q
real, external :: nofq, bofq
type (elements) :: ele
!     animalu form factor is given in Ry convert it to a.u.
  w0 = (nofq(ele,q) + bofq(ele,q)) / 2.0
end function w0
!! This calculation calculates the screened  HAA form factor
real function w(ele, q)
use defs
real ggvosko, q, qq, lindhard, vc, hartree
real, external :: w0
type (elements) :: ele
  qq = q / 2.0 / kf
  ggvosko = 0.5 * (q**2 / (q**2 + ksi * kf**2))
  lindhard = -mave * kf / pi**2 * (0.5 + (1.0 - qq**2) / 4.0 / qq * log(
(cont.)abs((1.0 + qq) / (1.0 - qq))))
  vc = 4.0 * pi / q**2
  hartree = 1.0 - vc * lindhard / (1.0 + vc * lindhard * ggvosko)
  w = w0(ele, q) / hartree
end function w
!! Ths routine find the vacancies in the md cell
subroutine findvac()
use commondefs
use vardecs
real dum
real rxij, ryij, rzij
real,external:: fcc
integer i, j
logical flag1
  call fcc(a, nbox_x, nbox_y, nbox_z)
  open(11, file = "fcc.txt")
    do i=1, nperf
      read(11,11) xini(i), yini(i), zini(i)
      11 format(f20.17, 4x, f20.17, 4x, f20.17)
    end do
  close(11)
  dum=1
  flag1=.false.
  do j=1,nperf
    do i=1,n+nwall
      rxij = xini(j) - atoms(i)%x
      ryij = yini(j) - atoms(i)%y
      rzij = zini(j) - atoms(i)%z
      if (dabs(rxij) .gt. tol) rxij = (dabs(rxij) - boxsize_x) * dsign
(cont.)(1.0,rxij)
      if (dabs(ryij) .gt. tol) ryij = (dabs(ryij) - boxsize_y) * dsign
(cont.)(1.0,ryij)
      if (dabs(rzij) .gt. tol) rzij = (dabs(rzij) - boxsize_z) * dsign
```

```
(cont.) (1.0,rzij)
        if (dsqrt(rxij**2+ryij**2+rzij**2).lt.(tol*dsqrt(3.0))) flag1=.
(cont.) true.
      enddo
      if (flag1 .eq. .false.) then
        atoms(n + nwall + dum)%x = xini(j)
        atoms(n + nwall + dum)%y = yini(j)
        atoms(n + nwall + dum)%z = zini(j)
        atoms(n + nwall + dum)%kind = -1
        dum = dum+1
      endif
      flag1 = .false.
    enddo
    nvac = dum-1
end subroutine findvac
!! This routine constructs an FCC cell from old data used in findvac
subroutine fcc(a, nx, ny, nz)
real a
real, dimension(4)::x0, y0, z0
integer ix, iy, iz, i, nx, ny, nz
real, allocatable:: x(:), y(:), z(:)
allocate(x(nx * ny * nz * 4), y(nx * ny * nz * 4), z(nx * ny * nz * 4))
  open(11, file = "fcc.txt", position = "rewind", status = "old")
  x0(1) = 0.0
  y0(1) = 0.0
  z0(1) = 0.0
  x0(2) = 0.0
  y0(2) = 0.5 * a
  z0(2)  =0.5 * a
  x0(3) = 0.5 * a
  y0(3) = 0.0
  z0(3) = 0.5 * a
  x0(4) = 0.5 * a
  y0(4) = 0.5 * a
  z0(4) = 0.0
  do iy = -1, ny - 2
    do ix = 0, nx - 1
      do iz = 0, nz - 1
        do i = 1, 4
          x(i) = ix * a + x0(i)
          y(i) = iy * a + y0(i)
          z(i) = iz * a + z0(i)
          write(11,10) x(i), y(i), z(i)
          10 format(f20.17, 4x, f20.17, 4x, f20.17)
        end do
      end do
    end do
  end do
  close(11)
end subroutine fcc
subroutine fcc2(a, nx, ny, nz)
real a
real, dimension(4)::x0, y0, z0
integer ix, iy, iz, i, nx, ny, nz
real, allocatable:: x(:), y(:), z(:)
allocate(x(nx * ny * nz * 4), y(nx * ny * nz * 4), z(nx * ny * nz * 4))
  open(11, file = "fcc.txt", position = "rewind", status = "old")
  x0(1) = 0.0
  y0(1) = 0.0
  z0(1) = 0.0
  x0(2) = 0.0
  y0(2) = 0.5 * a
  z0(2)  =0.5 * a
  x0(3) = 0.5 * a
  y0(3) = 0.0
  z0(3) = 0.5 * a
  x0(4) = 0.5 * a
  y0(4) = 0.5 * a
```

```
  z0(4) = 0.0
  do iy = -1, ny - 2
    do ix = 0, nx - 1
      do iz = 0, nz - 1
        do i = 1, 4
          x(i) = ix * a + x0(i)
          y(i) = iy * a + y0(i)
          z(i) = iz * a + z0(i)
          write(11,10) x(i), y(i), z(i)
          10 format(f20.17, 4x, f20.17, 4x, f20.17)
        end do
      end do
    end do
  end do
  close(11)
end subroutine fcc2
!! temperature control with velocity scaling
subroutine velscale()
use vardecs
use commondefs
real scalefactor, scalecount
integer i
  sumvel = 0.0
  do i = 1, n
    sumvel = sumvel + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz**2)
(cont.) * molwt(atoms(i)%kind)
  end do
 !scale velocities at scalestep intervals
  if (tstep .le. scalestep) then
    scalecount = mod(tstep, scaleint)
    if(scalecount.eq.0.0)then
      scalefactor = dsqrt(3.0 * n * bolz * temp / sumvel)
 !scale velocities to desired temperature
      do i = 1, np
        atoms(i)%vx = atoms(i)%vx * scalefactor
        atoms(i)%vy = atoms(i)%vy * scalefactor
        atoms(i)%vz = atoms(i)%vz * scalefactor
      end do
    end if
  end if
end subroutine velscale
!! andersen pressure control with velocity Verlet predictor step
subroutine anders1()
use vardecs
use commondefs
real sumro
integer i
  sumro = 0.0
  do i = 1, n
    atoms(i)%rox = atoms(i)%x / boxsize_x
    atoms(i)%roy = atoms(i)%y / boxsize_y
    atoms(i)%roz = atoms(i)%z / boxsize_z
    atoms(i)%rox1 = atoms(i)%vx / boxsize_x
    atoms(i)%roy1 = atoms(i)%vy / boxsize_y
    atoms(i)%roz1 = atoms(i)%vz / boxsize_z
    sumro = sumro + (atoms(i)%rox1**2 + atoms(i)%roy1**2 + atoms(i)%roz1
(cont.)**2) * molwt(atoms(i)%kind)
  enddo
  do i = 1, n
    atoms(i)%rox = atoms(i)%rox + tstp * atoms(i)%rox1 + 0.5 * tstp**2 *
(cont.) ((atoms(i)%fx / molwt(atoms(i)%kind)) / boxsize_x - 2.0 / 3.0 *
(cont.)atoms(i)%rox1 * volume1 / volume)
    atoms(i)%roy = atoms(i)%roy + tstp * atoms(i)%roy1 + 0.5 * tstp**2 *
(cont.) ((atoms(i)%fy / molwt(atoms(i)%kind)) / boxsize_y - 2.0 / 3.0 *
(cont.)atoms(i)%roy1 * volume1 / volume)
    atoms(i)%roz = atoms(i)%roz + tstp * atoms(i)%roz1 + 0.5 * tstp**2 *
(cont.) ((atoms(i)%fz / molwt(atoms(i)%kind)) / boxsize_z - 2.0 / 3.0 *
(cont.)atoms(i)%roz1 * volume1 / volume)
    atoms(i)%rox1 = atoms(i)%rox1 + 0.5 * tstp * ((atoms(i)%fx / molwt(
(cont.)atoms(i)%kind)) / boxsize_x - 2.0 / 3.0 * atoms(i)%rox1 * volume1 /
```

```
(cont.) volume)
     atoms(i)%roy1 = atoms(i)%roy1 + 0.5 * tstp * ((atoms(i)%fy / molwt(
(cont.) atoms(i)%kind)) / boxsize_y - 2.0 / 3.0 * atoms(i)%roy1 * volume1 /
(cont.) volume)
     atoms(i)%roz1 = atoms(i)%roz1 + 0.5 * tstp * ((atoms(i)%fz / molwt(
(cont.) atoms(i)%kind)) / boxsize_z - 2.0 / 3.0 * atoms(i)%roz1 * volume1 /
(cont.) volume)
   end do
   sumvel=0.0
   do i = 1, n
     atoms(i)%x = atoms(i)%rox * boxsize_x
     atoms(i)%y = atoms(i)%roy * boxsize_y
     atoms(i)%z = atoms(i)%roz * boxsize_z
     atoms(i)%vx = boxsize_x * atoms(i)%rox1
     atoms(i)%vy = boxsize_y * atoms(i)%roy1
     atoms(i)%vz = boxsize_z * atoms(i)%roz1
     sumvel = sumvel + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz**2)
(cont.) *molwt(atoms(i)%kind)
   enddo
   tempdum = (sumvel) /(3.0 * real(n) * bolz)
   pressure = ((n * bolz * tempdum) + (1.0 / 3.0) * totw) / volume
   volume = volume + tstp * volume1 + 0.5 * tstp**2 * (pressure - pset) /
(cont.) m
   volume1 = volume1 +  0.5 * tstp * (pressure - pset) / m
   boxsize_x = (volume**0.33333333) * bxr
   boxsize_y = (volume**0.33333333) * byr
   boxsize_z = (volume**0.33333333) * bzr
   sumvel = 0.0
   do i = 1, n
     atoms(i)%x = atoms(i)%rox * boxsize_x
     atoms(i)%y = atoms(i)%roy * boxsize_y
     atoms(i)%z = atoms(i)%roz * boxsize_z
     atoms(i)%vx = boxsize_x * atoms(i)%rox1
     atoms(i)%vy = boxsize_y * atoms(i)%roy1
     atoms(i)%vz = boxsize_z * atoms(i)%roz1
     sumvel = sumvel + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz**2)
(cont.) * molwt(atoms(i)%kind)
   enddo
end subroutine anders1
!! Andersen pressure control with velocity Verlet corrector step
subroutine anders2()
use vardecs
use commondefs
integer i
   do  i = 1, n
     atoms(i)%rox1 = atoms(i)%rox1 + 0.5 * tstp * ((atoms(i)%fx / molwt(
(cont.) atoms(i)%kind)) / boxsize_x - 2.0 / 3.0 * atoms(i)%rox1 * volume1 /
(cont.) volume)
     atoms(i)%roy1 = atoms(i)%roy1 + 0.5 * tstp * ((atoms(i)%fy / molwt(
(cont.) atoms(i)%kind)) / boxsize_y - 2.0 / 3.0 * atoms(i)%roy1 * volume1 /
(cont.) volume)
     atoms(i)%roz1 = atoms(i)%roz1 + 0.5 * tstp * ((atoms(i)%fz / molwt(
(cont.) atoms(i)%kind)) / boxsize_z - 2.0 / 3.0 * atoms(i)%roz1 * volume1 /
(cont.) volume)
   end do
   sumvel = 0.0
   do i = 1, n
     atoms(i)%x = atoms(i)%rox * boxsize_x
     atoms(i)%y = atoms(i)%roy * boxsize_y
     atoms(i)%z = atoms(i)%roz * boxsize_z
     atoms(i)%vx = boxsize_x * atoms(i)%rox1
     atoms(i)%vy = boxsize_y * atoms(i)%roy1
     atoms(i)%vz = boxsize_z * atoms(i)%roz1
     sumvel = sumvel + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz**2)
(cont.) * molwt(atoms(i)%kind)
   enddo
   tempdum = (sumvel) / (3.0 * real(n) * bolz)
```

```
   pressure = ((n * bolz * tempdum) + (1.0 / 3.0) * totw) / volume
   volume1 = volume1 +  0.5 * tstp * (pressure - pset) / m
end subroutine anders2
!! Berendsen pressure and temperature control
subroutine berendsen()
use vardecs
use commondefs
integer i
   sumvel = 0.0
   do i = 1,n
     sumvel = sumvel + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz**2)
(cont.) * molwt(atoms(i)%kind)
   end do
   tempdum = (sumvel) / (3.0 * real(n) * bolz)
   pressure = ((n * bolz * tempdum) + (1.0 / 3.0) * totw) / volume
   mu = (1 + (tstp / tau_p) * beta * (pressure - pset))**(1.0 / 3.0)
   lambda = (1 + (tstp / tau_t) * (temp / tempdum - 1))**(0.5)
   do i = 1, n
     atoms(i)%vx = atoms(i)%vx + tstp * atoms(i)%fx / molwt(atoms(i)%kind
(cont.))
     atoms(i)%vy = atoms(i)%vy + tstp * atoms(i)%fy / molwt(atoms(i)%kind
(cont.))
     atoms(i)%vz = atoms(i)%vz + tstp * atoms(i)%fz / molwt(atoms(i)%kind
(cont.))
     atoms(i)%vx = atoms(i)%vx * lambda
     atoms(i)%vy = atoms(i)%vy * lambda
     atoms(i)%vz = atoms(i)%vz * lambda
     atoms(i)%x = atoms(i)%x + tstp * atoms(i)%vx
     atoms(i)%y = atoms(i)%y + tstp * atoms(i)%vy
     atoms(i)%z = atoms(i)%z + tstp * atoms(i)%vz
     atoms(i)%x = atoms(i)%x * mu
     atoms(i)%y = atoms(i)%y * mu
     atoms(i)%z = atoms(i)%z * mu
   end do
   boxsize_x = boxsize_x * mu
   boxsize_y = boxsize_y * mu
   boxsize_z = boxsize_z * mu
   volume = volume * (mu**3)
end subroutine berendsen
!! Gear 5th order integration predictor step
subroutine predict(dt)
use vardecs
use commondefs
integer i
real c1, c2, c3, c4
real dt
   c1 = dt
   c2 = c1 * dt / 2.0
   c3 = c2 * dt / 3.0
   c4 = c3 * dt / 4.0
   do  i = 1, n
     atoms(i)%x = atoms(i)%x + c1 * atoms(i)%vx + c2 * atoms(i)%ax + c3 *
(cont.) atoms(i)%bx + c4 * atoms(i)%cx
     atoms(i)%y = atoms(i)%y + c1 * atoms(i)%vy + c2 * atoms(i)%ay + c3 *
(cont.) atoms(i)%by + c4 * atoms(i)%cy
     atoms(i)%z = atoms(i)%z + c1 * atoms(i)%vz + c2 * atoms(i)%az + c3 *
(cont.) atoms(i)%bz + c4 * atoms(i)%cz
     atoms(i)%vx = atoms(i)%vx + c1 * atoms(i)%ax + c2 * atoms(i)%bx + c3
(cont.) * atoms(i)%cx
     atoms(i)%vy = atoms(i)%vy + c1 * atoms(i)%ay + c2 * atoms(i)%by + c3
(cont.) * atoms(i)%cy
     atoms(i)%vz = atoms(i)%vz + c1 * atoms(i)%az + c2 * atoms(i)%bz + c3
(cont.) * atoms(i)%cz
     atoms(i)%ax = atoms(i)%ax + c1 * atoms(i)%bx + c2 * atoms(i)%cx
     atoms(i)%ay = atoms(i)%ay + c1 * atoms(i)%by + c2 * atoms(i)%cy
     atoms(i)%az = atoms(i)%az + c1 * atoms(i)%bz + c2 * atoms(i)%cz
     atoms(i)%bx = atoms(i)%bx + c1 * atoms(i)%cx
```

```
    atoms(i)%by = atoms(i)%by + c1 * atoms(i)%cy
    atoms(i)%bz = atoms(i)%bz + c1 * atoms(i)%cz
  end do
return
end subroutine predict
!!! This routine is the corrector stage of the Gear 5th order
(cont.) integration
subroutine correct(dt)
use vardecs
use commondefs
real axi,ayi,azi,corrx,corry,corrz
real c1,c2,c3,c4
real gear0,gear1,gear3,gear4
real cr,cv,cb,cc
real dt
integer i
  gear0 = 19.0 / 120.0
  gear1 = 3.0 / 4.0
  gear3 = 1.0 / 2.0
  gear4 = 1.0 / 12.0
  c1 = dt
  c2 = c1 * dt / 2.0
  c3 = c2 * dt / 3.0
  c4 = c3 * dt / 4.0
  cr = gear0 * c2
  cv = gear1 * c2 / c1
  cb = gear3 * c2 / c3
  cc = gear4 * c2 / c4
  do i = 1 , n
    axi = atoms(i)%fx / molwt(atoms(i)%kind)
    ayi = atoms(i)%fy / molwt(atoms(i)%kind)
    azi = atoms(i)%fz / molwt(atoms(i)%kind)
    corrx = axi - atoms(i)%ax
    corry = ayi - atoms(i)%ay
    corrz = azi - atoms(i)%az
    atoms(i)%x = atoms(i)%x + cr * corrx
    atoms(i)%y = atoms(i)%y + cr * corry
    atoms(i)%z = atoms(i)%z + cr * corrz
    atoms(i)%vx = atoms(i)%vx + cv * corrx
    atoms(i)%vy = atoms(i)%vy + cv * corry
    atoms(i)%vz = atoms(i)%vz + cv * corrz
    atoms(i)%ax = axi
    atoms(i)%ay = ayi
    atoms(i)%az = azi
    atoms(i)%bx = atoms(i)%bx + cb * corrx
    atoms(i)%by = atoms(i)%by + cb * corry
    atoms(i)%bz = atoms(i)%bz + cb * corrz
    atoms(i)%cx = atoms(i)%cx + cc * corrx
    atoms(i)%cy = atoms(i)%cy + cc * corry
    atoms(i)%cz = atoms(i)%cz + cc * corrz
  end do
return
end
!! velocity Verlet predictor step
subroutine verl1(dt)
use vardecs
use commondefs
real dt,dtsq2,dt2
integer i
  dt2   = dt / 2.0
  dtsq2 = dt * dt2
  do i = 1, np
    atoms(i)%x = atoms(i)%x + dt * atoms(i)%vx + dtsq2 * (atoms(i)%fx /
(cont.) molwt(atoms(i)%kind))
    atoms(i)%y = atoms(i)%y + dt * atoms(i)%vy + dtsq2 * (atoms(i)%fy /
(cont.) molwt(atoms(i)%kind))
    atoms(i)%z = atoms(i)%z + dt * atoms(i)%vz + dtsq2 * (atoms(i)%fz /
```

```
(cont.) molwt(atoms(i)%kind))
     atoms(i)%vx = atoms(i)%vx + dt2 * (atoms(i)%fx / molwt(atoms(i)%kind
(cont.)))
     atoms(i)%vy = atoms(i)%vy + dt2 * (atoms(i)%fy / molwt(atoms(i)%kind
(cont.)))
     atoms(i)%vz = atoms(i)%vz + dt2 * (atoms(i)%fz / molwt(atoms(i)%kind
(cont.)))
  end do
return
end subroutine verl1
!! velocity Verlet corrector step
subroutine verl2(dt)
use vardecs
use commondefs
real dt, dt2
integer i
  dt2 = dt / 2.0
  totkin = 0.0
  do i = 1, np
    atoms(i)%vx = atoms(i)%vx + dt2 * atoms(i)%fx / molwt(atoms(i)%kind)
    atoms(i)%vy = atoms(i)%vy + dt2 * atoms(i)%fy / molwt(atoms(i)%kind)
    atoms(i)%vz = atoms(i)%vz + dt2 * atoms(i)%fz / molwt(atoms(i)%kind)
    totkin = totkin + (atoms(i)%vx**2 + atoms(i)%vy**2 + atoms(i)%vz**2)
(cont.) * molwt(atoms(i)%kind)
  end do
  totkin = totkin * 0.5
return
end subroutine verl2
!! This routine calculates velocity autocorrelation function
subroutine vacf(name1, tstep, sstep, dtime, n, icf)
real vaf
integer i, j, n, step, nsample, tstep, sstep, icf
real vx0(n), vy0(n), vz0(n), x0(n), y0(n), z0(n), vx(n), vy(n), vz(n)
real x(n), y(n), z(n), v0(n), r0(n), dtime, time
character name1*(*)
character(len = len_trim(name1)) name2
  name2 = trim(name1)
  vaf = 0.0
  open(unit = 3, file = name2//".con")
  open(unit = 2, file  =name2//".vaf")
  read (3, *) step
  do i = 1, n
    read(3, 31) x0(i), y0(i), z0(i), vx0(i), vy0(i), vz0(i)
    31 format(1x, 6es15.7)
    v0(i) = dsqrt(vx0(i) * vx0(i) + vy0(i) * vy0(i) + vz0(i) * vz0(i))
    r0(i) = dsqrt(x0(i)**2 + y0(i)**2 + z0(i)**2)
    vaf = vaf + (v0(i) * v0(i)) / (v0(i)* v0(i))
  end do
  vaf  =vaf / real(n)
  time = real((step - (sstep + icf))) * dtime
  nsample = (tstep - sstep) / icf
  write (2, *) time, vaf
  do j = 1, nsample - 1
    vaf = 0.0
    read (3, *) step
    do i = 1, n
      read(3,32) x(i),y(i),z(i),vx(i),vy(i),vz(i)
      32 format(1x,6es15.7)
      vaf = vaf + (vx0(i) * vx(i) + vy0(i) * vy(i) + vz0(i) * vz(i)) / (
(cont.) v0(i)* v0(i))
    end do
    vaf = vaf / real(n)
    time = real((step - (sstep + icf))) * dtime
    write (2, *) time, vaf
  end do
  close(2)
end subroutine vacf
```