Fall 1-7-2011

# Charge Transfer in Deoxyribonucleic Acid (DNA): Static Disorder, Dynamic Fluctuations and Complex Kinetic.

Neranjan S. Edirisinghe Pathirannehelage
*Georgia State University*

# CHARGE TRANSFER IN DEOXYRIBONUCLEIC ACID (DNA): STATIC DISORDER, DYNAMIC FLUCTUATIONS AND COMPLEX KINETIC.

by

NERANJAN S. EDIRISINGHE PATHIRANNEHELAGE

Under the Direction of Vadym Apalkov

## ABSTRACT

The fact that loosely bonded DNA bases could tolerate large structural fluctuations, form a dissipative environment for a charge traveling through the DNA. Nonlinear stochastic nature of structural fluctuations facilitates rich charge dynamics in DNA. We study the complex charge dynamics by solving a nonlinear, stochastic, coupled system of differential equations. Charge transfer between donor and acceptor in DNA occurs via different mechanisms depending on the distance between donor and acceptor. It changes from tunneling regime to a polaron assisted hopping regime depending on the donor-acceptor separation. Also we found that charge transport strongly depends on the feasibility of polaron formation. Hence it has complex dependence on

temperature and charge-vibrations coupling strength. Mismatched base pairs, such as different conformations of the G・A mispair, cause only minor structural changes in the host DNA molecule, thereby making mispair recognition an arduous task. Electron transport in DNA that depends strongly on the hopping transfer integrals between the nearest base pairs, which in turn are affected by the presence of a mispair, might be an attractive approach in this regard. I report here on our investigations, via the $I-V$ characteristics, of the effect of a mispair on the electrical properties of homogeneous and generic DNA molecules. The $I-V$ characteristics of DNA were studied numerically within the double-stranded tight-binding model. The parameters of the tight-binding model, such as the transfer integrals and on-site energies, are determined from first-principles calculations. The changes in electrical current through the DNA chain due to the presence of a mispair depend on the conformation of the G・A mispair and are appreciable for DNA consisting of up to 90 base pairs. For homogeneous DNA sequences the current through DNA is suppressed and the strongest suppression is realized for the G(anti)・A(syn) conformation of the G・A mispair. For inhomogeneous (generic) DNA molecules, the mispair result can be either suppression or an enhancement of the current, depending on the type of mispairs and actual DNA sequence.

INDEX WORDS: Deoxyribonucleic acid, Non equilibrium Greens' function, Model hamiltonian, Stochastic differential equations, Polaron, Parallel and distributed computing, Tight binding model

# CHARGE TRANSFER IN DEOXYRIBONUCLEIC ACID (DNA): STATIC DISORDER, DYNAMIC FLUCTUATIONS AND COMPLEX KINETIC.

by

NERANJAN S. EDIRISINGHE PATHIRANNEHELAGE

A Dissertation Submitted in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

in the College of Arts and Sciences

Georgia State University

2010

CHARGE TRANSFER IN DEOXYRIBONUCLEIC ACID (DNA): STATIC DISORDER, DYNAMIC FLUCTUATIONS AND COMPLEX KINETIC.

by

NERANJAN S. EDIRISINGHE PATHIRANNEHELAGE

Committee Chair: Vadym Apalkov

Committee: Gennady Cymbalyuk

Mukesh Dhamala

Brian Thoms

Donald Hamelberg

Electronic Version Approved:

Office of Graduate Studies

College of Arts and Sciences

Georgia State University

December 2010

# DEDICATION

To my ever loving parents, wonderful wife Nileesha and my sweet angel Nethni.

ACKNOWLEDGMENTS

My gratitude goes to my parents for allowing me to become me. Humble admiration for my first teacher who thought me how to distinguish right and wrong, some thirty years ago. Reverence for the spiritual leaders who thought to move forward until success irrespective of the obstacles. Heartiest appreciation for my advisor Dr. Vadym Apalkov for his guidance, endurance and all the support during my time at Georgia State University. Special thanks to Dr. Jaroslav Klc at GSU End User Tools Engineering for his precious support during the technical difficulties and Dr. Gennady Cymbalyuk for his constructive advice in intricate periods. I deeply appreciate the graduate committee members, Dr. Gennady Cymbalyuk, Dr. Mukesh Dhamala, Dr. Brian Thoms and Dr. Donald Hamelberg. I would also like to thank all of the faculty and staff and friends in the Georgia State University Department of Physics & Astronomy who have helped me along with my academic career. Finally many thanks to my wife Nileesha and daughter Nethni for their patience, support and all the love that one could ever have.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

# Chapter 1. The Fascinating Helical: The Deoxyribonucleic acid

Mammalian brain, the most fascinating creation existing in nature has attained an unattainable computational power at very low power dissipations. In-spite of modern super computers has achieved peta-flop processing power; it is far away from achieving the supreme state of the human brain. As an example, power consumption and dissipation in such system is a major problem. Nano scale electronic has gained tremendous attention in recent years owing to increasing demand for the speed and power efficiency of modern electronics. Recent developments in medical technologies also demand for the smaller and smaller materials. But development of structures of size of one billionth of a meter is a challenging task. On the other hand, if some material possesses self assembly, it would be certainly preferred as a candidate for nano scale materials. In this prospective Deoxyribonucleic acid (DNA), the signature fingerprint which carries genetic code of organisms from generation to generation, has maintained its leading candidacy as a material for nano electronic devices. Nevertheless its properties have been a subject of wildly debated. Formulation of suitable theoretical model has become an important requirement towards development of DNA electronics and in general, to understand the charge transport properties of DNA. The structural complexity imposes computational limitation towards using highly detail models such as molecular dynamics and abinito methods whereas coarse-grained models also needs to be developed in such a way it retains all necessary information facilitating realistic description. In this dissertation I address the question of developing minimal coarse-grained model which represents highly sophisticated DNA dimmers.

## Section 1.01 Structure of the DNA

Even a half a century after the discovery of its helical structure by Watson and Crick (Watson J.D. and Crick 1953, Wilkins M.H.F. 1953) with the aid of crystallographic images made by Franklin and Gosling (Franklin R. and Gosling 1953) and three decades from first postulation of electrical conductivity along the chain by Eley and Spivey (Eley and Spivey 1962), DNA remains a fascinating research subject, due to its highly complex structural dynamics and interactions with environments. As illustrated in Figure 1, DNA is a helical structure made out of two separate helixes. In principle this structure can be partitioned into two main regions, the backbone; the outer skeleton of the DNA, and the inner conducting channel, made out of well stacked base-pairs. The backbone of the DNA strand is made from alternating phosphate and sugar residues. The sugar in DNA is 2-deoxyribose, which is a five-carbon sugar (pentose). Two adjacent sugar residues in a single strand DNA are connected by phosphate groups by phosphodiester bonds between the third and fifth carbon atoms of adjacent sugar rings (Figure 2). The lone electron pair at the phosphate group makes the DNA a negatively charged entity. Each sugar residue is connected to the third residue called DNA bases, which can either be Adenine, Cytosine, Guanine or Thymine (Figure 3). These four bases are classified into two broad categories based on their structural composition. Adenine and Guanine are composed of fused five- and six-member heterocyclic aromatic rings called purines, while Cytosine and Thymine are composed of six-member rings called pyrimidine. The complementary arrangement in the aromatic ring favors specific self assembly patterns between Cytosine (C), Guanine (G) and Adenine (A), Thymine (T). The strength of the bonding between a given base pair is determined by the number of hydrogen bonds present in-between participating bases. Cytosine

(C) and Guanine (G) share three hydrogen bonds whereas Adenine (A) and Thymine (T) share only two (Figure 4).

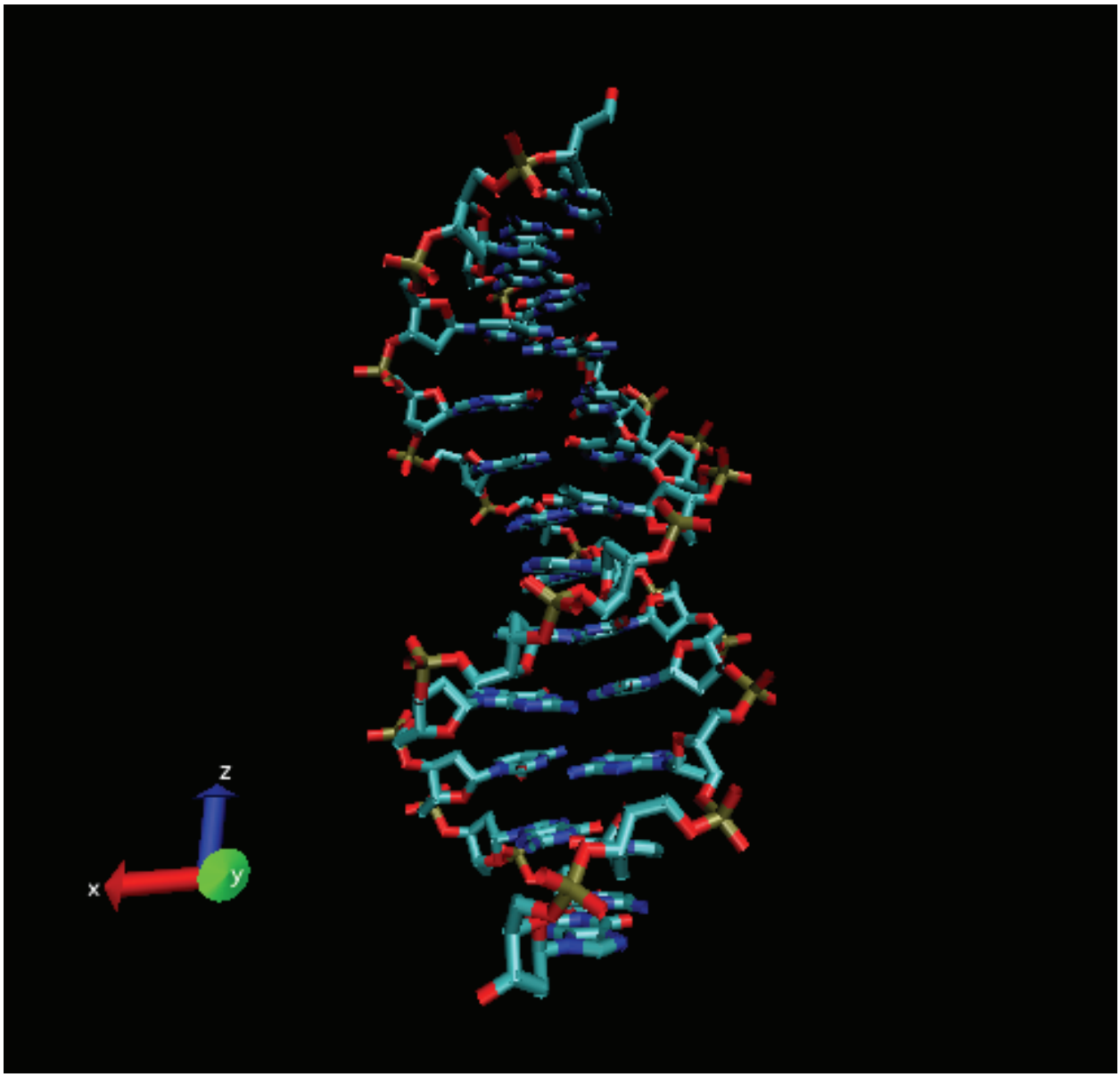


Figure 1: Helical Structure of DNA: DNA helical consists of two single strands DNA. Two strands are anti parallel to each other and have complimentary bases. Complementary bases are connected to their complements by hydrogen bonds.

Weak; yet sufficiently strong hydrogen bonds hold two single strand DNAs to form a double strand DNA which exists in living organisms. This special arrangement makes double strand DNA structure a three dimensional helical with approximately 10 base pairs for complete revolution. The thickness of the double strand is around 20 Å, whereas the distance between two adjacent base pairs is around 3.4 Å (Barbi, Cocco and Peyrard 1999, Smith, Cui and Bustamante 1996). Due to the specific directionality of the single strand DNA and the selective complementary hydrogen bonds, two stands in the double strand DNA is anti parallel to each other. Unlike solid crystals, the soft skeleton of the DNA is vulnerable to large amplitude oscillations due to thermal forces.

## Section 1.02 Electronic structure

The fact, that DNA base pairs are aromatic entities, i.e. organic compounds containing planer, unsaturated, benzene type ring structure, shoreup the speculation that DNA might support long distance charge transport through well delocalized $\pi$ orbital. The $p_z$ orbital of the DNA base pairs, which is perpendicular to the base pair plane, could form rather delocalized $\pi$ bonding and $\pi^*$ anti-bonding molecular orbital. The energy gap between $\pi$ bonding and $\pi^*$ anti-bonding molecular orbitals is around 4 eV (Helgren E. et al. 2001). However unlike crystalline structures, the biological DNA is not a periodic entity. This limits the existence of extended bands throughout the DNA. On the other hand largest ionization potential difference between two isolated base pairs, which is observed between Guanine and Thymine, is about 0.6 eV. This value exceeds the estimated coupling between highest occupied molecular orbital and lowest unoccupied molecular orbital of neighboring base pairs, leading to expectation of Anderson localization (Anderson 1958) .

The strength of the transfer integral is determined by the extent of $p_z$ orbital overlap between neighboring base pairs.

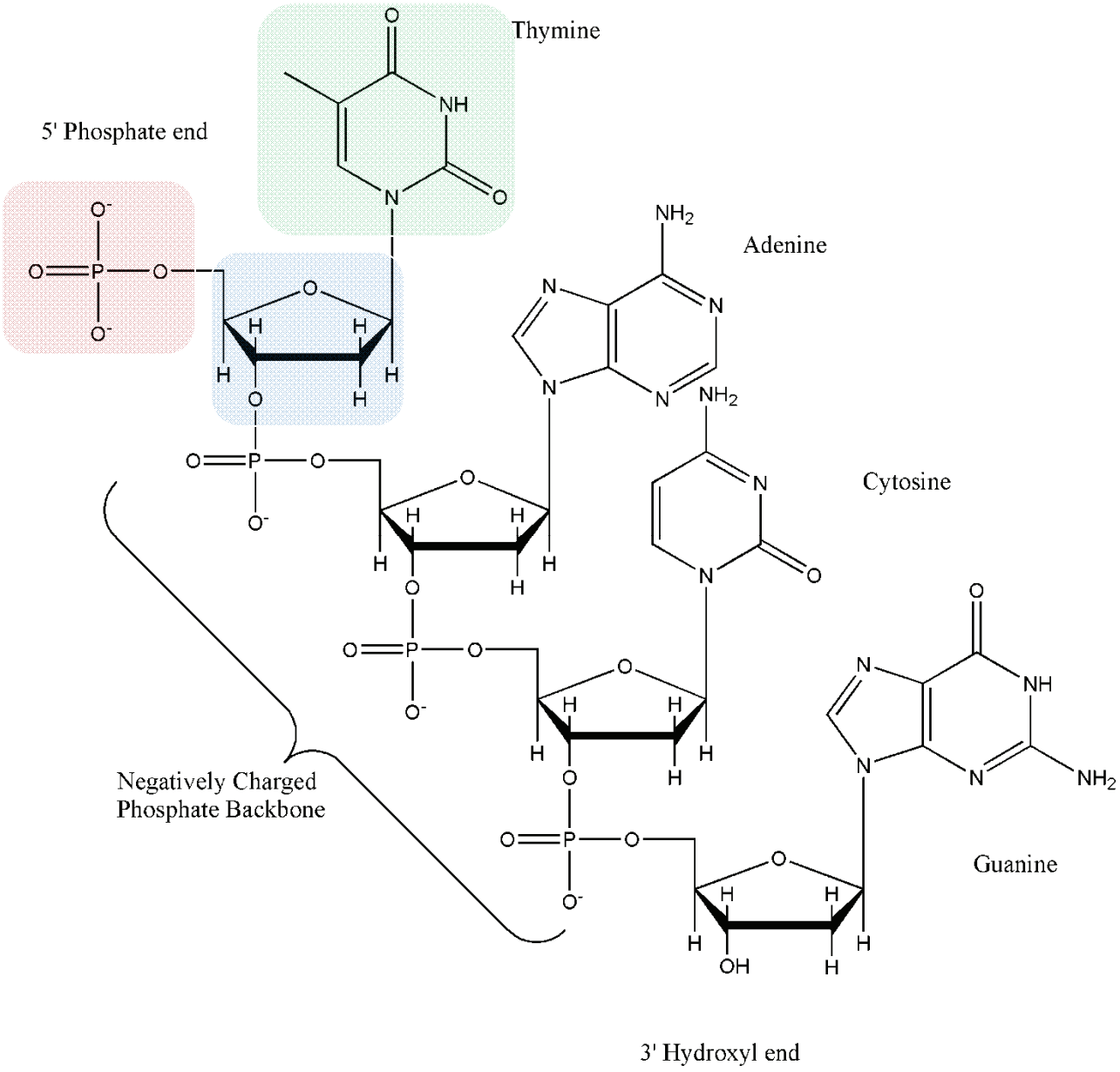


Figure 2: Structural arrangement of DNA base pairs: Each base in the single strand DNA is connected to a sugar group at 5th carbon atom of the sugar group. The two adjacent sugar groups are connected to each other by phosphate groups.

The relative positions of the $p_z$ orbital are determined by the twist angle and the relative space between two adjacent base pairs.

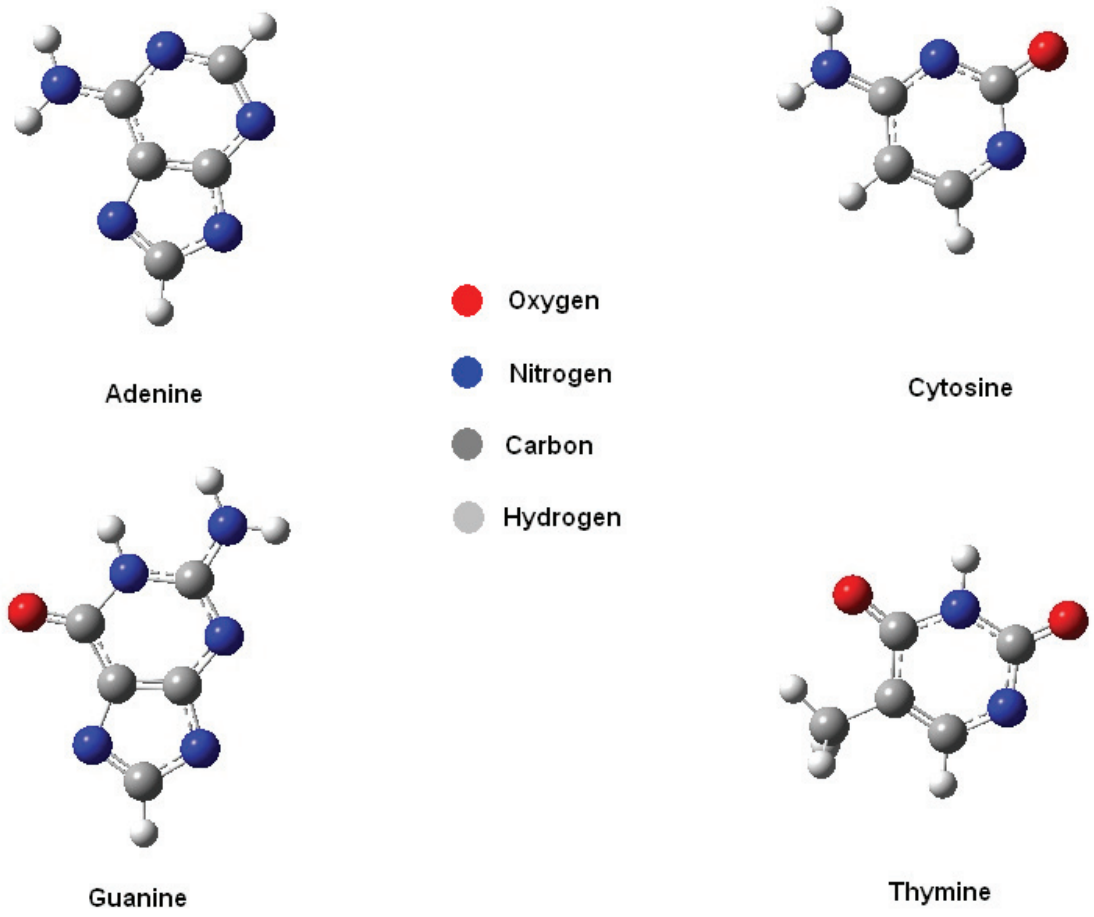


Figure 3: DNA bases: Adenine and Guanine are categorized as purines whereas Thymine and Cytosine are categorized as pyrimidine based on its structural composition.

The root-mean square displacement of the base pair at room temperature is about 0.3-0.4 Å (Matthew A. Young 1997). This is one tenths of the equilibrium distance between base pairs and order of magnitude larger than that of the crystals. Thermal fluctuation changes twist angle, on average, by as much as $8^{\circ}$ (Matthew A. Young 1997). The corresponding change of the transfer integral could be as large as 0.1 eV. Such large transfer integral change could introduce time

dependent local state into the system which could intern facilitate instantaneous charge transport through the DNA.

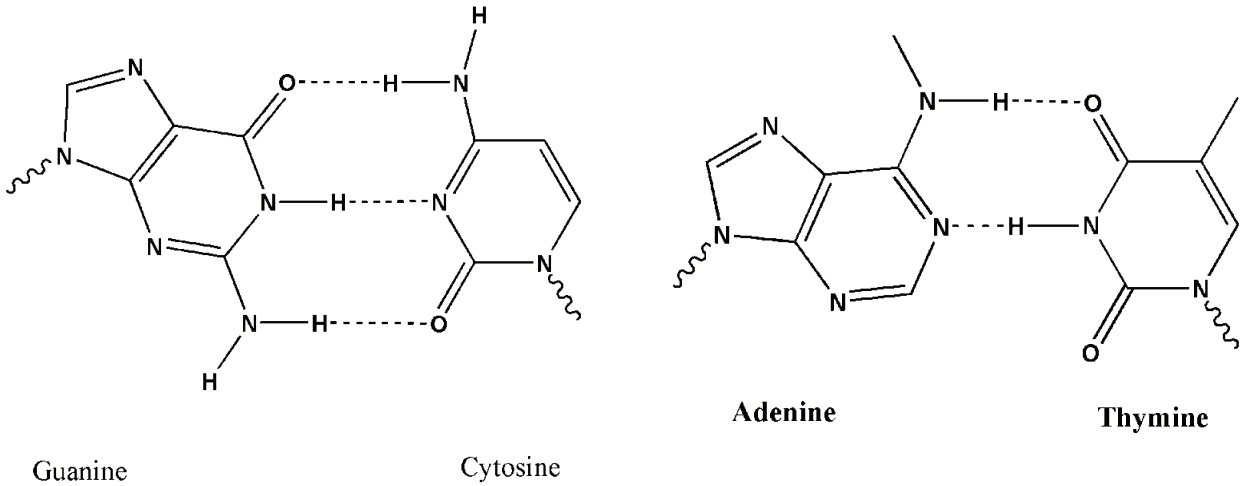


Figure 4 : Hydrogen bonding between DNA WC bases: G-C shares 3 hydrogen bonds, whereas A-T only shares two.

Another important fact that determines the structure of the DNA is the charge neutrality of DNA. DNA bases are hydrophobic, whereas negatively charged back-bone prefers interaction with cations and water molecules. As a result the stability of the DNA structure is critically influenced by the environment. For example at an environment with five to ten water molecules per base pair A-DNA structure is preferred whereas if the number of water molecule per base pair exceeds thirteen then B-DNA is preferred.

# Chapter 2. DNA modeling

With recent improvements of computational technologies it is becoming feasible to model large molecules using first principles. However using of first principle methods with DNA is still a challenging task due to the problem listed below. A successful utilization of the first principle methods on electronic structure calculation relies on the availability of a periodic structure. The presence of periodic structure facilitates the use of Bloch theorem. This enables us to perform most of the computation in the frequency domain. However a soft molecule like DNA barely possesses a periodic structure. For this reason, there still exist severe computational constrains for any all atom first principle calculation on DNA. On the other hand all atom molecular dynamic simulations are in general limited by the memory availability. We have found that for an accurate description of charge dynamics in the DNA it may be necessary to study a DNA strand with at least 512 base pairs. All atom molecular dynamics of 512 base pair is challenging as it would account for thousands of atoms. For this reason the development of a good coarse grained model is a necessity. In this dissertation I try to develop an accurate coarse grained model for DNA and study its properties.

## Section 2.01 Physical Structure

### (a) Peyrard & Bishop Model

As discussed above, the two strands of DNA are held together by weak hydrogen bonds. This flexible hydrogen bond facilitates large oscillations of magnitude in the DNA. In fact under normal conditions DNA undergoes complex three dimensional oscillations. However our main emphasis in this dissertation is to study the charge transport through the DNA and we try to make our model as simple as possible yet providing meaningful results. As a starting point to

study charge transport through DNA we adopt a model originally developed by M. Peyrard and A.R. Bishop (Maniadis et al. 2003, Kalosakas, Ngai and Flach 2005) to study DNA degradation. In this model each base in the DNA WC base pair is treated as a point mass. Each base is allowed to oscillate around the center of mass while the center of mass remains stationary. In this quasi one dimensional model only transverse motion of the base pair are considered. In general the longitudinal displacements are several orders of magnitude smaller than that of transverse displacements and hence, as a starting point, longitudinal displacements can be safely omitted from the consideration.

As shown in the Figure 5, in this model each base in the DNA is treated as a point mass. The double strand DNA is composed of two one dimensional chains connected by hydrogen bonds between complementary base pairs. The neighboring bases in a single strand are coupled to each other through sugar phosphate backbone. The vibration along the DNA chain can be considered as harmonic. Even though this could be an over simplification of the system, this treatment had produced the essential properties of DNA degradations. However the dynamics along the transverse direction cannot be treated using simple harmonic approximation since more than one counteracting forces present along the transverse direction. Moreover the hydrogen bonds connecting two bases at different strand become extremely stretched when double helix opens up locally. This type of extremely large displacement, rather nonlinear, cannot be accounted by the harmonic approximation. Peyrard and Bishop, in their treatment of DNA degradation (Maniadis et al. 2003), have used Morse potential to describe nonlinear displacement associated with hydrogen bond stretching along the transverse direction. Indeed in the transverse direction there are two opposing forces. The attractive bonding force of hydrogen bond and the repulsive forces between phosphate groups which is partially screened by the solvent and ions presence in the

media (Linak and Dorfman 2010). The Morse potential accounts for the average potential due to these two counter forces.

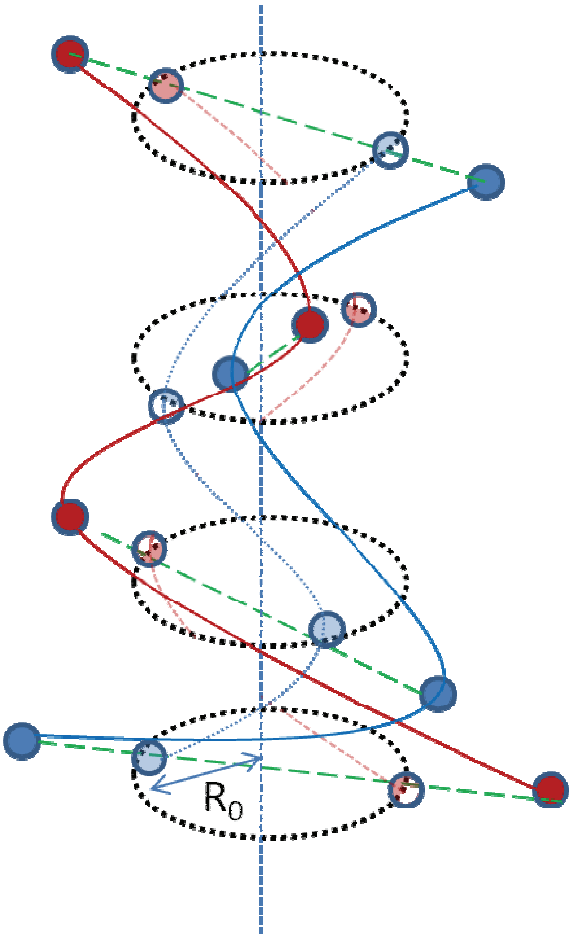


Figure 5 : Bishop model of DNA: each base in the DNA is represented by a point mass. Two bases in a base pair are connected by an elastic rod and free to move along the transverse direction. But in this model any other motion is prohibited. The light colored circuls reprecents the equilibrium positions where as the dark colored circuls relrecents the displaced position of the bases.

By denoting the displacement of base pair from its equilibrium position by $w_n$ for strand one and $v_n$ for strand two, where n represents the $n^{th}$ position along a given strand, the Lagrangian for the motion of a point masses (DNA WC base) can be written as;

$$L = \sum_n \frac{1}{2} m(\dot{w}_n^2 + \dot{v}_n^2) + \frac{1}{2} k[(w_n - w_{n-1})^2 + (v_n - v_{n-1})^2] + V(w_n - v_n) \qquad (1)$$

Where V is the Morse potential given by;

$$V(w_n - v_n) = D\{exp[-a(w_n - v_n)] - 1\}^2 \quad (2)$$

By introducing new variables $x_n = (u_n + v_n)/\sqrt{2}$ and $y_n = (u_n - v_n)/\sqrt{2}$ along the normal coordinate this equation can be simplify into;

$$L = L(x) + L(y) = \sum_n \left\{\frac{p_n^2}{2m} + \frac{1}{2}k(x_n - x_{n-1})^2\right\}$$

$$+ \sum_n \left\{\frac{q_n^2}{2m} + \frac{1}{2}k(y_n - y_{n-1})^2 + D\left[exp\left(-a\sqrt{2}y_n\right) - 1\right]^2\right\} \quad (3)$$

In the above equation $p_n$ and $q_n$ are the momentum of the particle with respect to the variables $x_n$ and $y_n$. Here m is the mass of the base. The advantage of this description is that the Lagrangian in equation (3) which depends only on the variable x is now decoupled from the stretching part (i.e. Morse potential does not depend on the x variable). This portion of the Lagrange can be ignored from further consideration.

With this assumption the final form of the Lagrangian can be written as;

$$L = \sum_n \left\{\frac{q_n^2}{2m} + \frac{1}{2}k(y_n - y_{n-1})^2 + D\left[exp\left(-a\sqrt{2}y_n\right) - 1\right]^2\right\} \quad (4)$$

The influence of structural fluctuation on charge transport properties is mainly caused by the changes in onsite energy, which is a function of relative distance between two base pairs. It indeed depends on the difference in the stretching of the two bases in the same base pair, i.e. y.

(b) Nonlinear stretching in DNA

Later on it has been shown that the stacking interactions between DNA bases are indeed functions of not merely single base, but pair of adjacent base pairs. The observation that the decrease in stacking interaction with DNA degradation, in the degradation process DNA opens up locally, breaking hydrogen bonds between base pair and altering the electron structure of the individual bases, suggest that stretching interaction could depend on the interaction between two bases rather individual base. The harmonic assumption used in the previous model deficit the ability to represent such behavior adequately. This has been overcome by replacing the harmonic potential with an-harmonic potential given by the following equation.

$$W(y_n, y_{n-1}) = \frac{k}{2}\left(1 + \rho e^{-\beta(y_n + y_{n-1})}\right)(y_n - y_{n-1})^2 \qquad (5)$$

The term $k/2\left(1 + \rho e^{-\alpha(y_n + y_{n-1})}\right)$ in the new potential extend the stretching interaction into two base pairs. When either base pair is stretched this term reduces the effective potential from $k/2(1 + \rho)$ to $k/2$. With this new potential a base pair in the vicinity of an open site has lower vibration frequency which reduces the contribution to the free energy.

(c) Beyond Peyrard & Bishop Model

Even though the model described above sufficiently describes the DNA degradation, we have found that it is inadequate to describe the charge transport properties through the DNA. The Peyrard & Bishop model only considers the in plane transverse motion. But starting with the speculation that twist motion may also play an important role in DNA charge transport as the charge transfer integral are very sensitive to the changes in twist angle we added the twist angle to the existing model. Due to the symmetry of the problem it will be convenient to work in

spherical coordinate system. The coordinate system that we have used is defined as follows. The radial distance is measured from the helical axis of the DNA. Azimuth is measured with respect to the equilibrium twist angle.

Similar to the previous model, in the present model each base is also treated as a point mass. But now two point masses are allowed to rotate around the helical axis simultaneously. However any independent rotations are still forbidden, which means that, always two bases in the base pair can be connected by a straight line. Each neighboring bases in a given strand are considered to be connected by an elastic rigid rods. The equilibrium length of this rod is assumed to be constant among different base pairs. Longitudinal motion is not considered due to the same reasons as in the above model.

The Lagrangian for the system can then be written in the following form. First term in the equation represents the kinetic energy of the particle. The Morse potential in the second term represents the hydrogen bond interaction and the repulsion between phosphate backbones. The third term account for the rigidity of the rod and the last term is added to the Lagrangian to ensure the helical structure of the system.

$$\mathcal{L} = \sum_n (m\dot{r}_n^2 + m r_n^2 \dot{\varphi}_n^2) - D\left(e^{-\alpha(r_0 - R_0)} - 1\right)^2$$
$$- \sum_n K\left(\sqrt{h^2 + r_{n-1}^2 + r_n^2 - 2r_{n-1}r_n cos(\varphi_n - \varphi_{n-1})} - l\right)^2$$
$$- \sum_n G_0(\varphi_{n+1} + \varphi_{n-1} - 2\varphi_n)^2 \qquad (6)$$

We have introduced the dimensionless time as $t' = \sqrt{\frac{D\alpha^2}{m}}t$ and the dimensionless length as $r' = \alpha r_n$. The equation of motion is then computed following the same procedure as given at (Barbi et al. 1999).

$$\ddot{\phi}_n = K_{\phi\phi}(\phi_{n+1} + \phi_{n-1} - 2\phi_n) - G(\phi_{n+2} + \phi_{n-2} - 4\phi_{n+1} - 4\phi_{n-1} + 6\phi_n) + \frac{K_{y\phi}}{2}(y_{n+1} - y_{n-1})$$
$$- \frac{2}{R_0}\dot{y}_n\dot{\phi}_n - \frac{2}{R_0}y_n\ddot{\phi}_n - \frac{2}{R_0^2}y_n\dot{y}_n\dot{\phi}_n - \frac{1}{R_0^2}y_n^2\ddot{\phi}_n \quad (7)$$

$$\ddot{y}_n = \left(1 + \frac{y_n}{R_0}\right)\frac{1}{R_0}\dot{\phi}_n^2 - \left(y_n - \frac{3}{2}y_n^2 + \frac{7}{6}y_n^3\right) - K_{yy}(y_{n+1} + y_{n-1} - 2y_n) - \frac{K_{y\phi}}{2}(\phi_{n+1} - \phi_{n-1}) \quad (8)$$

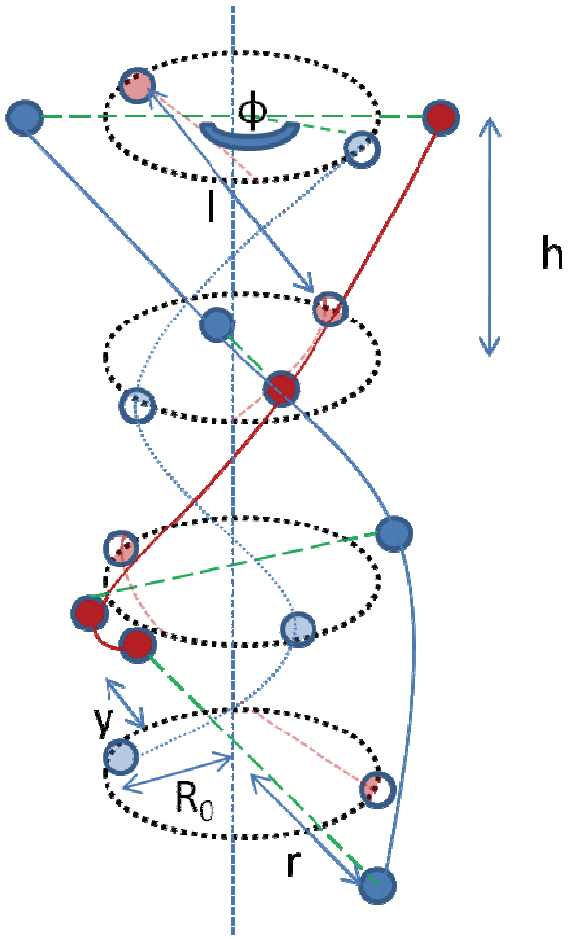

Figure 6 : the schematic diagram of a DNA. Each base is treated as a point mass. Two adjacent bases in the same strand are connected by an elastic rod of length l. The two bases in the same base pair are allowed to vibrate in plane transversely. In addition in this model the DNA is allowed to have angular vibrations. The twist angle is measures as the deviation from the equilibrium twist angle.

## Section 2.02 Electronic Structure

In order to model electronic structure of the DNA one must investigate the relative contribution from the different parts of DNA towards the electrical properties. The two strands of double strand DNA winds up each other to form the helical structure. As discussed in the previous chapters DNA can be divided into two main parts, the backbone and the base pair stacking. The connection between base pairs occurs through the backbone sugar phosphate group. However it has been shown that the charge transport through DNA mainly occurs through the well stacked $\pi$ orbital of the bases (Boon and Barton 2002, Murphy et al. 1993, Kelley et al. 1999). Due to this reason backbone can be neglected in treating electronic properties of the DNA. However backbone of the DNA can influence charge transport properties indirectly (Cuniberti et al. 2002, Maciá and Roche 2006). The hydrophobic DNA bases of DNA strand reside inside the helix while hydrophilic sugar phosphate backbone faces the solution, usually polar water molecule and ions. The electrophilic backbone, due to electro negative phosphate group, favors the presence of positively charge ions. The presence of positively charge ions, such as $Na^+$ and $K^+$ ions, change the local environment around the DNA due to electrostatic forces (Voityuk 2005). The main outcome of such interactions is the change in the onsite potential of the backbone sites. These interactions can lead to new arrangements of the base pairs, as bases of the two stands are coupled with weak hydrogen bonds which are free to move. In the presence of external cations and water molecules, DNA sugar group tends to move with respect to the base pair and tries to find new conformations which minimize the system energy. Such

movement changes the effective overlap between the electron clouds of base pair and that of sugar group altering the charge transfer integral. Therefore in the presence of environment ions and the solvent molecules, DNA tries to minimize its energy by altering its conformation. This in turn changes the charge transport properties through DNA. Effectively the interaction of sugar group with polar water molecules and ions changes the electron cloud at the sugar group which in turn can change the electron cloud at the base (Voityuk 2005). This changes the onsite energy of the DNA bases. The following onsite energy renormalization (Joe, Lee and Hedin 2010) can be used to incorporate the effect of the sugar group into the onsite energy of the bases.

$$\varepsilon_K(E) = \varepsilon_K + t_\perp + \frac{\{\tau_K(E)\}^2}{E - \sigma_K} \tag{9}$$

$$\tau_\propto(E) = \tau_\propto + \frac{\varepsilon_\propto(E-\sigma_\propto)}{\tau_\propto} \tag{10}$$

In the above equations $\varepsilon_K$ represent the onsite energy of the bases. The K can be either G, A, T or C. The $t_\perp$ is the inter-base coupling constant, $\sigma_\propto$ is the onsite energy of the sugar group at K base, $\sigma_K$ is the onsite energy of the isolated base, $\tau_\propto$ is the coupling between the base and the sugar group. As it can be seen in the Equation (9) the onsite energy of the base is now depends on the energy of the electron.

However in most calculations, for simplicity, it is possible to neglect the effects due to the backbone. In such situations the bases in the single strand DNA is assumed to couple through electrostatic interaction of stacked $\pi$ orbital of the bases.

(a) Tight binding model

This allows us to model the charge transport through single DNA using tight binding type Hamiltonian. The electronic Hamiltonian for $2p_z$ orbitals within a tight binding model with the nearest neighbors interactions only is given as

$$\mathcal{H} = \sum_i \varepsilon_i c^\dagger{}_i c_i + \sum_{i,j} t_{i,j} c_i^\dagger c_j + h.c. \quad (11)$$

In equation (11) $c^\dagger{}_i$ ($c_i$) represents the creation (annihilation) operator for a charge at the site i. The $\varepsilon_i$ represents the onsite energy. The second term represents the coupling between different sites. $t_{i,j}$ is the coupling strength between site i and site j.

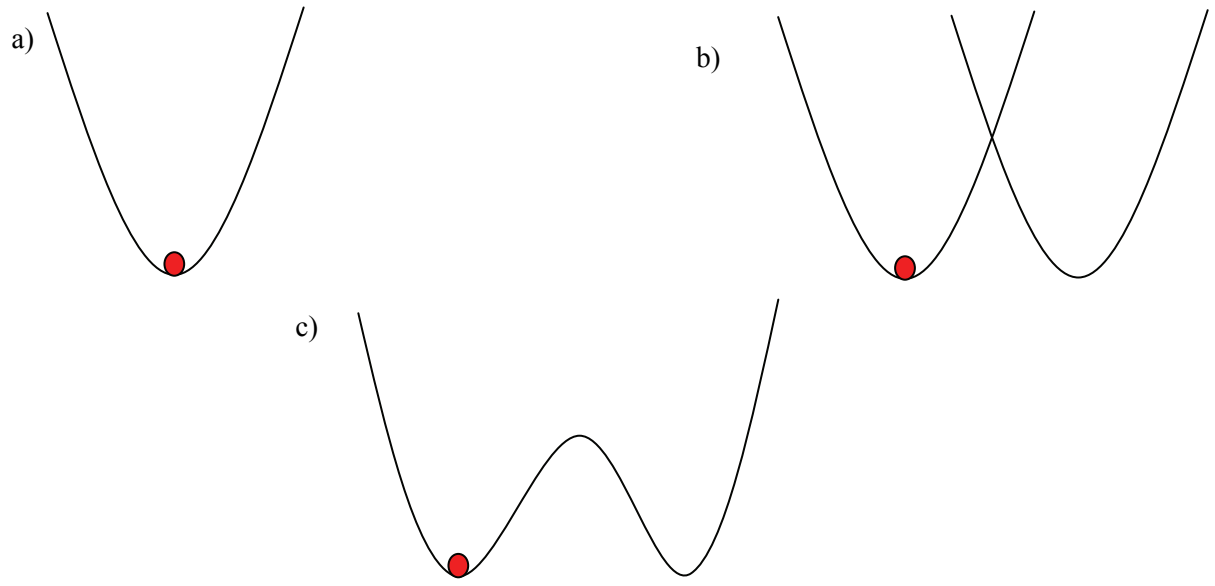


Figure 7 : Tight binding formulation: a) potential profile due to nucleus. b) For two nuclei brought closer the resultant potential profile can be described by the onsite energy and the coupling between them. c) Potential actually experienced by the particle. If overlapping is small then this profile can me modeled by tight binding type Hamiltonian.

Within this framework an electron at a given base pair experiences the attraction from all the nucleus in the base and the repulsion from all other electrons. The potential due to the resultant

interaction determines the behavior of the electron at this particular base. However in tight binding model the resultant of all the atoms are replaced by a point and electron is assumed to move under the influence of resultant potential Figure 7. If a second WC base pair brought into the vicinity of the first WC base pair, then the energy profile experience by the electron at the first base pair will change. The magnitude of change depends on the relative coupling of the two potential profiles. In general the coupling is determined by how well these two potential profiles overlap with each other. If two WC base pairs are very close to each other, the distortion may be too strong so that the original energy profiles of individual bases may not describe the new energy profile adequately. In such situation it is impossible to utilize tight binding formulation. However, in DNA WC base pairs are sufficiently apart from each other, separated by average distance of 3.4A, such that it is possible to approximate the resultant energy profile of the DNA as a weighted sum the individual energy profiles and tight binding formulation can be justified.

In the Equation (11) we consider only one energy level. However in a molecule like DNA charge transport could occurs through many energy levels. In fact as many as 3000 valence orbitals may be needed to account charge transport precisely. Nevertheless a tradeoff has to be made between the accuracy needed and the computational complexity of the study. In the current dissertation we use different levels of coarse graining, as the objectives in different simulations differ from each other.

Figure 8 illustrate different possibilities of coarse graining. As mentioned above, Equation (11) discusses the situation with charge transport through the single strand using only one energy band. In the case of double strand DNA, one could take into account only one band per base. i.e. as seen in the Figure 8 c) the HOMO and LUMO bands usually reside on different bases in the double strand DNA. As an example for poly (G) –poly (C) double strand homogeneous DNA it

can be assumed that HOMO level resides on G whereas LUMO resides on C. This allows one to construct a Hamiltonian described in Equation (12)

$$\mathcal{H}_{DNA} = \sum_{i,K,\sigma} \varepsilon_{K,i} c^{\dagger}{}_{K,i,\sigma} c_{K,i,\sigma} + \sum_{i,K,\sigma} t_K c^{\dagger}_{K,i,\sigma} c_{K,i+1,\sigma} + \sum_{i,\sigma} t_{\perp} c^{\dagger}_{H,i,} c_{L,i,\sigma} + h.c. \qquad (12)$$

In the Equation (12) $c^{\dagger}_{K,i,\sigma}(c_{K,i,\sigma})$ represent creation (annihilation) of an electron with spin $\sigma = \pm 1/2$ at site i. The subscript K represents HOMO or LUMO energy levels. $\varepsilon_{K,i}$ is the onsite energy at site *i* for an electron in either HOMO or LUMO level. $t_K$ is the transfer integral between two nearest neighbor. $t_{\perp}$ is the transfer integral between HOMO and LUMO energy levels between the same base pair.

In a more sophisticated treatment one could include the sugar phosphate backbone into the consideration ( Figure 7 ).

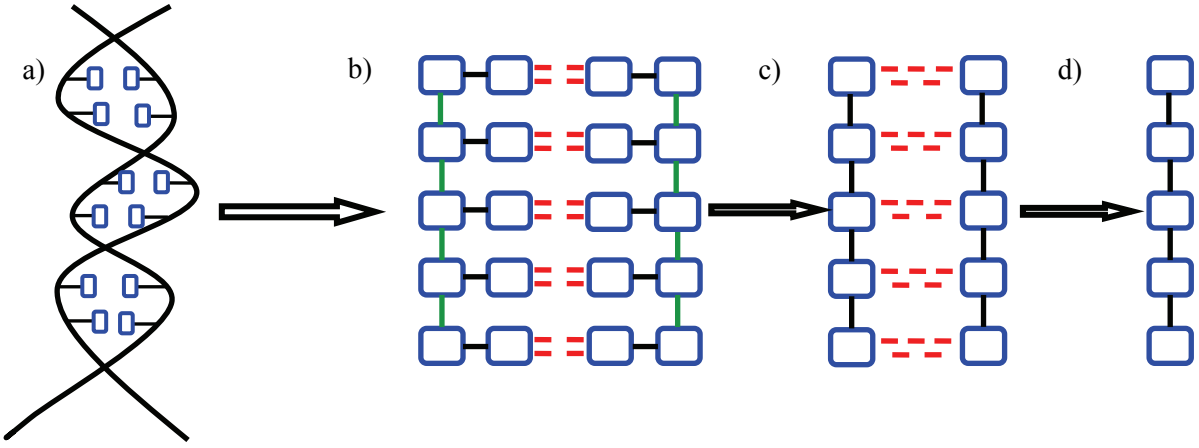


Figure 8 : Course graining of DNA: DNA contains two helical chains which run anti parallel to each other (a). The most sophisticated level of coarse graining this could be represented by 4 chains. Outer two chains represent the sugar phosphate back-bone and inner two chains represent the base pair overlap (b). The next level would be only to consider base pairs, completely neglecting the sugar phosphate backbone as it seldom contribute to the charge migration along the DNA (c). The simplest possible coarse graining is to treat DNA as a one dimensional chain by treating each base pair with single onsite energy and transfer integral between two adjacent base pairs (d).

The Hamiltonian for such system can be written as

$$H_{DNA} = \mathcal{H}_{DNA} + \sum_{i,\propto} \sigma_{\propto} b_{i,\propto}^{+} b_{i,\propto} - \sum_{i,\propto} \tau_{\propto} \left( b_{i,\propto}^{+} c_{i,\propto} + h.c. \right) \quad (13)$$

In this Hamiltonian first two terms describes charge dynamics of bases. Where $c_i^+$ $(c_i)$ is the creation (annihilation) operator of an electron at the DNA base and $\varepsilon_i$ represent the onsite energy of the base.

The first term in the Hamiltonian ($\mathcal{H}_{DNA}$) is given be Equation (12). The second term describes the sugar group and the final term describes the coupling between the sugar group and the base pair. $b_{i,\propto}^{+}(b_{i,\propto})$ represents the creation (annihilation) operators for a electron at i$^{th}$ site and $\propto$ represent either G or C for G-C base pair.

Section 2.03 Towards conducting DNA:

Although the first postulation that DNA could conduct charges for a long distance has been made as early as 1962, first experimental observation doesn't happen until 30 years later. The first experimental evidence that DNA might support long range charge transport came through an experiment conducted by Barton and co-workers in 1993 (Murphy et al. 1993, Ohshiro and Umezawa 2006). In this experiment they have reported that charge transport between DNA-intercalated transient metal complexes. Barton and co-workers have demonstrated that photo induced oxidization could travel as far as 40 Å in time short as tenth of a nano second (Hall, Holmlin and Barton 1996). The discovery that known damage to DNA could be healed by a photo excited rhodium intercalated molecule situated at 16 base pairs away has further enhanced the notion that DNA could support long range charge transport (Dandliker, Holmlin and Barton 1997) . Furthermore they have shown that electron migrating through DNA could be suppressed by causing a damage to the DNA chain (Kelley et al. 1999). But at the Barton's group almost all

observations were made using electrochemical methods. The fact that all these results show weak distance dependence (Murphy et al. 1993, Wan et al. 2000) of charge transport eventually leads to speculate that DNA could act as a "molecular wire" (Berlin, Burin and Ratner 2000).

However over the years many contradicting results have been published by the various research groups. These experimental results are amazingly different, covering all possible outcomes ranging from insulator (de Pablo et al. 2000, Braun et al. 1998, Storm et al. 2001), semiconductor (Porath et al. 2000), Ohmic (Fink and Schonenberger 1999, Cai, Tabata and Kawai 2000, Yoo et al. 2001), and superconductor(Kasumov et al. 2001).

As discussed above, in order to investigate charge transport properties through DNA researchers around the world used different methods which can be categorized into two broad categories. The first branch includes the measurement of IV characteristic of the DNA by connecting DNA to two conducting electrodes at the two ends (Fink and Schonenberger 1999, Sonmezoglu et al. 2010). The other branch consists of measuring charge transfer rate between donor and acceptor intercalators placed on the DNA strand (Kelley et al. 1999, Joy and Schuster 2005). These two categories of experiment provide distinct yet useful insight on the mechanism of charge transfer through DNA. Computationally these two experiments can be modeled using non-equilibrium Green function methods (Tsukamoto et al. 2009, Edirisinghe et al. 2010) and solving time dependent system of equations respectively (Hiroaki and Kazumoto 2010). In Chapter 3 I will formulate the computational framework which can be used to simulate I-V characteristic through the DNA. In Chapter 4 I will discuss the computational aspects of the charge transport between donor and acceptors sites using nonlinear stochastic system of equations.

Chapter 3. I-V Characteristic of DNA

Section 3.01 Current through DNA connected to two gold electrodes:

Most of the transport measurements through DNA are carried out by attaching two electrodes to the two ends of the DNA. First step toward the mathematical formulation of such systems is to develop a Hamiltonian describing the whole system, i.e. DNA and electrodes. The Hamiltonian for a DNA connected to two conducting electrode can be written as (Berlin et al. 2000, Roche 2003, Zhu, Kaun and Guo 2004);

$$\mathcal{H} = \mathcal{H}_{DNA} + \mathcal{H}_{leads} + \mathcal{H}_{DNA-leads} \tag{13}$$

In the Equation (13) $\mathcal{H}_{leads}$ describes the leads and the last term describes the interaction between leads and the DNA whereas $\mathcal{H}_{DNA}$ refers to the full Hamiltonian of the DNA. In the current discussion, the DNA is modeled using its HOMO and LUMO molecular orbital as shown in Figure 9. In such formulation different pathways are available in which charge can move through the DNA. For example charge can first jump to the HOMO level from left electrode and move along the extended orbital and jump to the right electrode, or else it can first jump to the HOMO level at the left electron and then jump to the LUMO levels and move along the extended LUMO molecular orbital. In order to model such behavior we have used the Hamiltonian presented in Equation (14).

$$\mathcal{H}_{DNA} = \sum_{i,K,\sigma} \varepsilon_{K,i} c^{\dagger}_{K,i,\sigma} \quad c_{K,i,\sigma} + \sum_{i,K,\sigma} t_K c^{\dagger}_{K,i,\sigma} c_{K,i+1,\sigma} + \sum_{i,\sigma} t_{\perp} c^{\dagger}_{H,i,\sigma} c_{L,i,\sigma} + h.c. \tag{14}$$

In this Hamiltonian; the operators $c_k^{\dagger}$ ($c_k$ ) represents the creation (annihilation) of an electron with momentum k and spin ± $1/2$. The first term accounts for the onsite energy. The second term describes the charge transfer between neighboring HOMO (LUMO) levels. Where $t_K$ , K=H(L) , is the charge transfer integral between the HOMO (LUMO) levels of neighboring

bases. The third term describes the charge transfer between the HOMO and LUMO levels of the same base pairs (Edirisinghe et al. 2010).

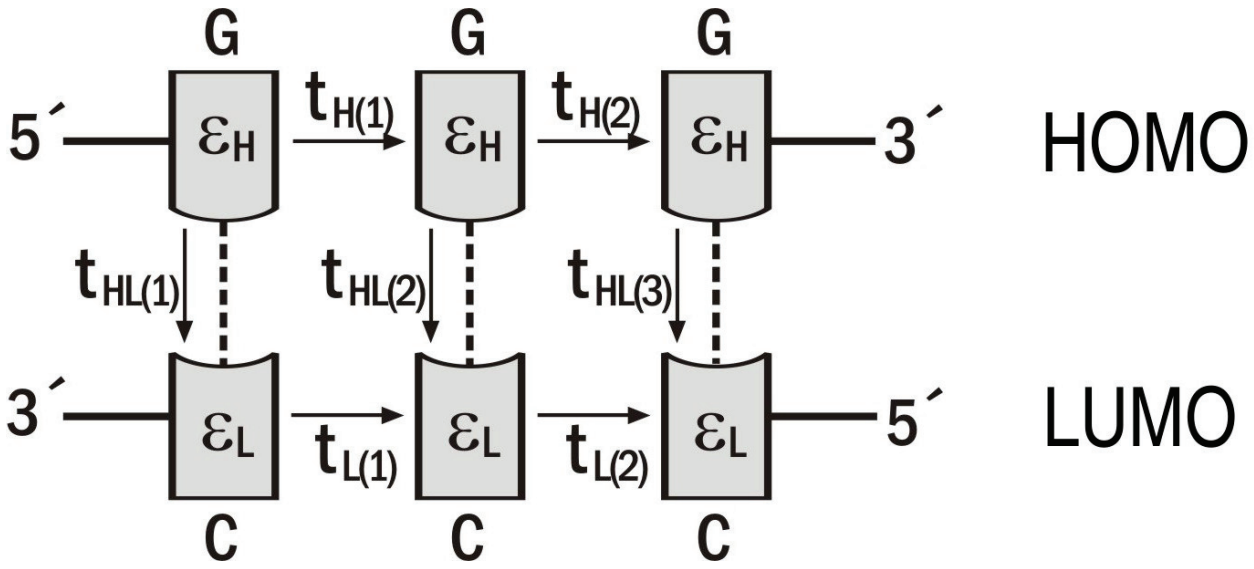


Figure 9 : Tight binding formulation for homogenous DNA base pairs. As HOMO level is assumed to be resides in the G whereas LUMO levels is assumed to be reside in C

Section 3.02 Charge transfer integrals

Table 1 presents the charge transfer integrals between different molecular orbital. For example charge transfer integral between the HOMO orbital at two adjacent G-C base pairs is -0.133 eV whereas the charge transfer integral between LUMO orbital is -0.041 eV, which is significantly smaller than HOMO for G-C base pairs. This suggests the charge transport through the adjacent G-C base pairs is mainly occurs through the HOMO level rather than LUMO levels. On the other hand the charge transfer integrals between the HOMO and LUMO levels of the same base pair is rather small. For example charge transfer integral between HOMO and LUMO of G-C base pair is -0.005 eV (Table 1). A different interpretation for transfer integrals can be givens in following form.

Table 1 : Charge transfer integrals between DNA regular Watson-Crick base pairs..

| | $V_{h_{m,n}}$ | | $V_{l_{m,n}}$ | | $V_{h_{m,n}}$ | |
|---|---|---|---|---|---|---|
| | G-C | A-T | G-C | A-T | G-C | A-T |
| G-C | -0.133 | -0.102 | -0.05 | -0.063 | -0.041 | 0.067 |
| A-T | -0.218 | -0.011 | -0.001 | 0.054 | 0.066 | -0.075 |

Say as shown in Figure 10 the two different atomic orbital hybridized to form a molecular orbit. This will form a bonding molecular orbital which is lower than the energy of the subscribing atomic orbital and anti- bonding orbital which is higher in energy than the subscribing orbital. The splitting between the energy of bonding orbital and anti-bonding orbital depends on the relative overlap between subscribing orbital. If the subscribing orbital has same energy and the orientation match each other then the energy split will be higher. i.e. it will form strong bonding orbital and the energy of that bonding orbital will be much less than the original orbital. However if the energy of subscribing orbital are different, it will not form good bonding and the energy splitting will also small. This energy splitting between bonding and anti bonding orbital are equal to two times the transfer integral between the two atomic orbital. Along this line we can anticipate that, since HOMO and LUMO energy levels have very different energies the coupling between them is very small whereas the transfer integrals between HOMO level of neighboring bases are relatively large.

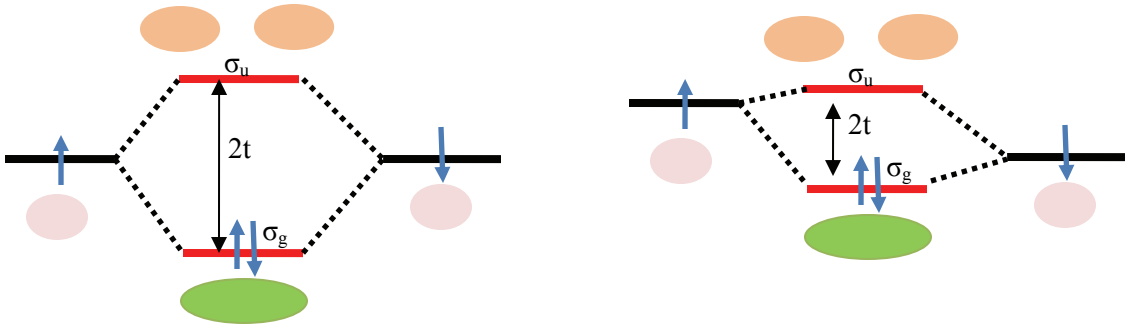


Figure 10 : Another interpretation to transfer integrals

Section 3.03 Non-equilibrium green function approach

The transport experiment has shown that in order to produce reproducible results one must need to form a chemical bonding between contact and the DNA. (Porath et al. 2000, Yoo et al. 2001, Hwang et al. 2002, Xu et al. 2004). On the other hand if the coupling is too strong one has to consider a state which is belonged to DNA-Contact system rather than to the DNA along (Emberly and Kirczenow 1998). But it is generally assumed the coupling between contact and DNA to be weak (Endres, Cox and Singh 2004, Porath, Cuniberti and Di Felice 2004, Ratner 1999) .

In the weak coupling regime the non equilibrium green functions can be used to model the current through the system consists of two weakly coupled electrodes and the DNA.

The stationary current at left electrode can be as;

$$J = \frac{ie}{\hbar}\sum_{kn}\left(V_k\langle c_k^\dagger d_n\rangle - V_{kn}^*\langle d_n^\dagger c_k\rangle\right) \tag{15}$$

In this equation $c_k^\dagger(c_k)$ represents the creation (annihilation) operators for a charge at electrode with momentum k whereas $d_n^\dagger\ (d_n)$ represents creation (annihilation) operators for electron at

the site n of the DNA. The coupling of a DNA chain to the leads is introduced through hopping between the HOMO and LUMO states of the first base pair of DNA and the left contact and between the HOMO and LUMO states of the N$^{th}$ base pair and the right contact. $V_{1,k}$ and $V_{n,k}$ (see Equation 15) are the hoping integrals between right electrode and first base pair and left electrode and N base pair respectively. In general a little is known about the strength of the coupling constant.

Using the definition of the Green function, i.e. $G^{<}_{n,k} = i\langle C^{\dagger}_{k} d_n \rangle$ , Equation (15) can be transformed into;

$$J = \frac{e}{\hbar}\sum_{kn}\int_{-\infty}^{\infty}\frac{d\omega}{2\pi}\left[V_{kn}G^{<}_{kn}(\omega) - V^{*}_{kn}G^{<}_{kn}(\omega)\right] \tag{16}$$

By writing Dysons equation for $G^{<}_{kn}(\omega)$ and $G^{<}_{nk}(\omega)$ current can be written in terms of the properties of the lead.

$$G^{<}_{kn}(\omega) = \sum_{m} V_{km}\left[g^{t}_{k,k}(\omega)G^{<}_{m,n}(\omega) - g^{<}_{k,k}(\omega)G^{\bar{t}}_{m,n}(\omega)\right] \tag{17}$$

$$G^{<}_{nk}(\omega) = \sum_{m} V^{*}_{k,m}\left[g^{<}_{k,k}(\omega)G^{t}_{n,m}(\omega) - g^{\bar{t}}_{k,k}(\omega)G^{<}_{n,m}(\omega)\right] \tag{18}$$

Where;

$$g^{<}_{k,k}(\omega) = 2\pi i f_L(\omega)\delta(\omega - \varepsilon_k) \tag{19}$$

$$g^{>}_{k,k}(\omega) = -2\pi i[1 - f_L(\omega)]\delta(\omega - \varepsilon_k) \tag{20}$$

Finally; using the definitions of advance and retarded Green functions this expression can conveniently be written as;

$$G^{r} = G^{t} - G^{<} \tag{21}$$

$$G^{a} = G^{t} - G^{>} \tag{22}$$

$$J = \frac{ie}{\hbar}\sum_{n,m}\int d\varepsilon\rho(\varepsilon)V_n(\varepsilon)V_m^*(\varepsilon)\{f_L(\varepsilon)[G_{n,m}^r(\varepsilon) - G_{n,m}^a(\varepsilon)] + G_{n,m}^<\} \quad (23)$$

It should be noted that in Eq. (21) current is expressed in terms of local properties of DNA strand, by means of retarded and advanced green functions, and distribution functions of the leads. Similar expression can be obtained for the right leads also. Using Eq. (17) - Eq. (22) current can be simplify into Eq. (23)

$$G^< = if_L(\varepsilon)G^r\Gamma^L G^a + if_R(\varepsilon)G^r\Gamma^R G^a \quad (24)$$

$$G^r - G^a = -iG^r(\Gamma^L + \Gamma^R)G^a \quad (25)$$

Finally using Equation (24) and Equation (25) current through the DNA connected to two electrodes can be written as Equation (26);

$$J = \frac{e}{h}\int d\varepsilon[f_L(\varepsilon) - f_R(\varepsilon)]Tr\{G^a\Gamma^R G^r\Gamma^L\} \quad (26)$$

Where, $\Gamma_{n,m}^L = 2\pi\sum\rho(\varepsilon)V_n(\varepsilon)V_m^*(\varepsilon)$ , $\Gamma_{n,m}^R$has similar definition. Note that and $\Gamma_{n,m}^L$ and $\Gamma_{n,m}^R$ are 2N*2N level-width matrices, determining the coupling of the DNA states through the continuous states of the right and left contacts, respectively. Tr{} correspond to the trace over the matrix. $f_L(\varepsilon)$ and $f_R(\varepsilon)$ are the Fermi–Dirac distribution functions for the left and right contacts, respectively;

The retarded $G_r$ and advanced $G_a$ Green functions of electrons in the DNA chains computed using equation of motion approach.

$$G^{r/a}(\varepsilon) = \left(\varepsilon - H_t \pm \frac{1}{2}(\Gamma^R + \Gamma^L)\right)^{-1} \quad (27)$$

In the matrix for the Hamiltonian describing the system can be written as Equation (22)

As described above the Green function method can be used to shrink the infinite dimensional system into a finite dimensional system (System becomes infinite as the contact is a conductor.). In this formulation the Hamiltonian for the system we study can be illustrate by the Figure 11. As can be seen the effect of the leads is now included in the onsite energy term at the two contacting sites (Anantram, Lundstrom and Nikonov 2008).

## Section 3.04 A qualitative overview on charge transport through DNA

Figure 11 provide a qualitative description of the system. As seen in the Figure 11 the molecular orbital of each base pair hybridized to form extended minibands across the whole molecule. The overall energy of the minibands is determined by energy of the participating molecular orbital. And the width of the mini band is determined by the energy split cased by the effective overlapping of these molecular orbital, in other words by the transfer integrals between molecular fragments.

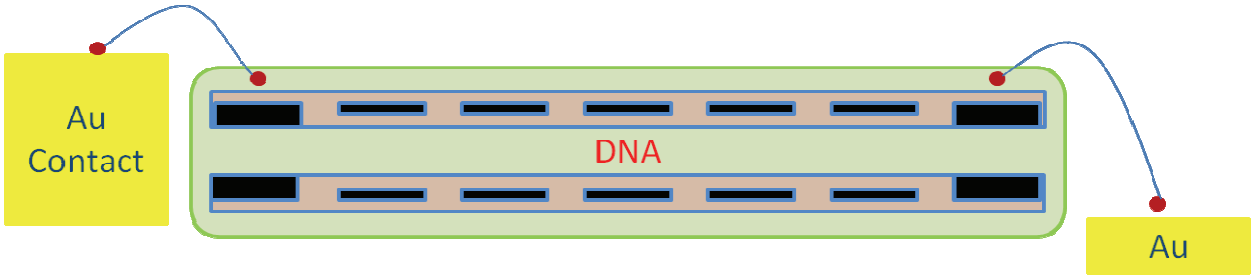


Figure 11 : Schematic diagram of DNA connected to the gold contacts. The well localized molecular orbital form extended energy minibands which facilitate long range charge transport through DNA.

In a molecule like DNA, a large number of base pairs contribute to form the extended bands. The width of this band depends on the relative energy match and the overlap between the contributing molecular orbital. If the contributing orbital are at same energy and the effective overlap is large, the resulting bands will have larger band width whereas if it has different energies and poor overlap then the resulting bans will have small band width. In this respect one

would imagine that DNA bands would have large band width since the contributing $p_z$ orbitals are stacked in such a way that facilitate good overlapping. However in the case of DNA, the distances between two bases are relatively large. In the direction of the helical axis this distance is around 3.4 Å . This results in significantly reduce the overlap between the $p_z$ and result in thin bands

Section 3.05 What determines the current through the DNA

In the direction of the helical axis this distance is around 3.4 Å whereas in the direction perpendicular to the helical axis this distance is around 10 Å. On the other hand the energy difference between base pairs along the direction perpendicular to the axis is also large. Since base pair consist of different types of bases. This results in relatively small intra strand transfer integral. On the other hand along the helical axis, the distance between base pairs are relatively small and there can be same type of bases adjacent to each other since the connection comes through the sugar phosphate backbone. Due to this reason the inter strand transfer integral are significantly large compared to that of intra strand. However the charge transfer integrals between the base pairs are in the range of meV. This results in relatively thin energy bands. These thin energy bands are highly sensitive to the energy profile of the DNA and the changes in transfer integrals. For this reason charge transport through DNA is subject to the effects due to surrounding media, defects such as mispair, temperature of the sample etc… The strength of charge transport through the DNA connected to two leads mainly determines by the shape of the equilibrium Eigen functions of the DNA. As DNA is at the equilibrium before connecting to the leads and the coupling to the contact is assumed to be weak, the deviation of non equilibrium wave function from the equilibrium Eigen function can assumed to be small. This allows us to use Eigen function of the system to gain qualitative insight into the system at its non equilibrium situations. As shown in the Figure 13 a) if the Eigen function of the system is well delocalized

across the whole DNA, and then the charge density at the edge is significantly large. In such situation there is finite probability that a charge at the contact move to the DNA and move across the DNA through the state defined by the wave function. However if the wave function of the system is localized as shown in the Figure 13 b), then the charge density at the edge of the DNA is small. This leads to small current through the system. In summary the current through DNA depends on different factors. In principle the existence of well delocalized molecular orbital is requires to have good charge conductance through DNA.

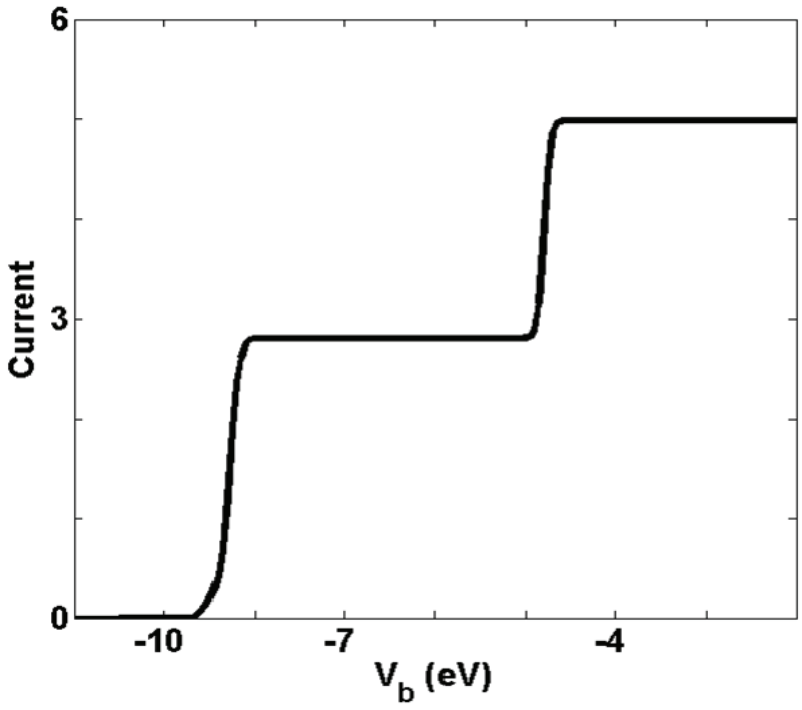


Figure 12 : I-V characteristic of homogeneous poly(G)-poly(C) DNA molecule. The steps represent the current through different mini bands. Energy profile determines the positions of the steps.

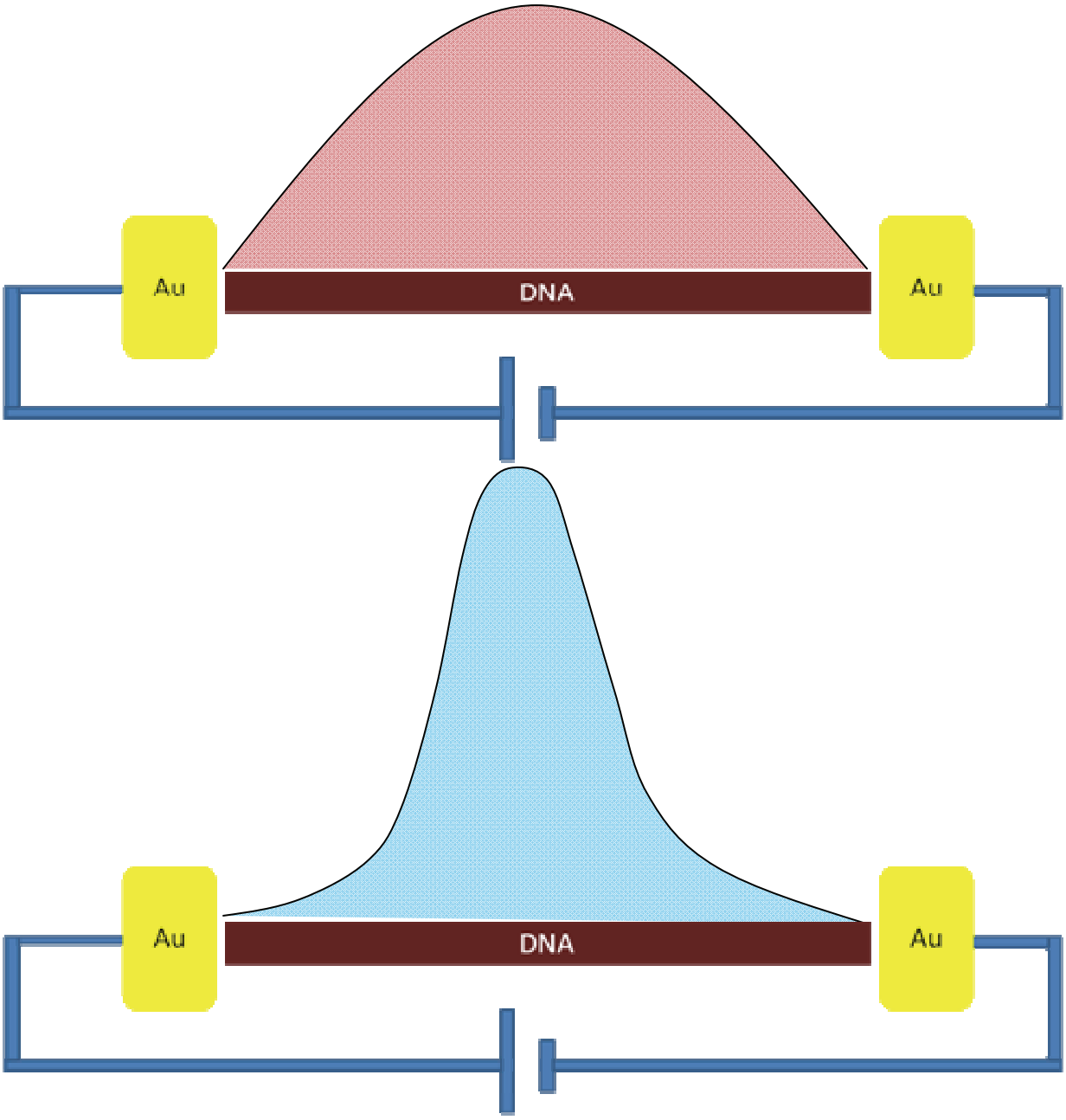


Figure 13 : Current through the DNA depends on its Eigen function. Is Eigen function of the system is well delocalized as shown in figure a) the finite charge density at the edge of the wave function makes finite overlap between the charge density of the contact. Which intern facilitates high current. On the other hand if the wave function if localize the small charge density at the edges will give rise to poor overlap between the charge density of the contact resulting poor current.

Section 3.06 Homogeneous poly (G) – poly (C) DNA

The Figure 12 illustrates the I-V characteristic of poly(G) –ploy(C) homogeneous DNA sequence. For homogeneous DNA sequence energy levels of each base pair align each other forming good energy band which span from one end to other. The homogeneity will also allow good orbital overlap. This facilitates relatively wide energy band width facilitating large current. The prominent two steps represent the current through HOMO and LUMO mini bands (Edirisinghe et al. 2010, Hodzic and Newcomb 2007, Xu, Endres and Arakawa 2007). The wave function for such system has properties of free particle wave function.

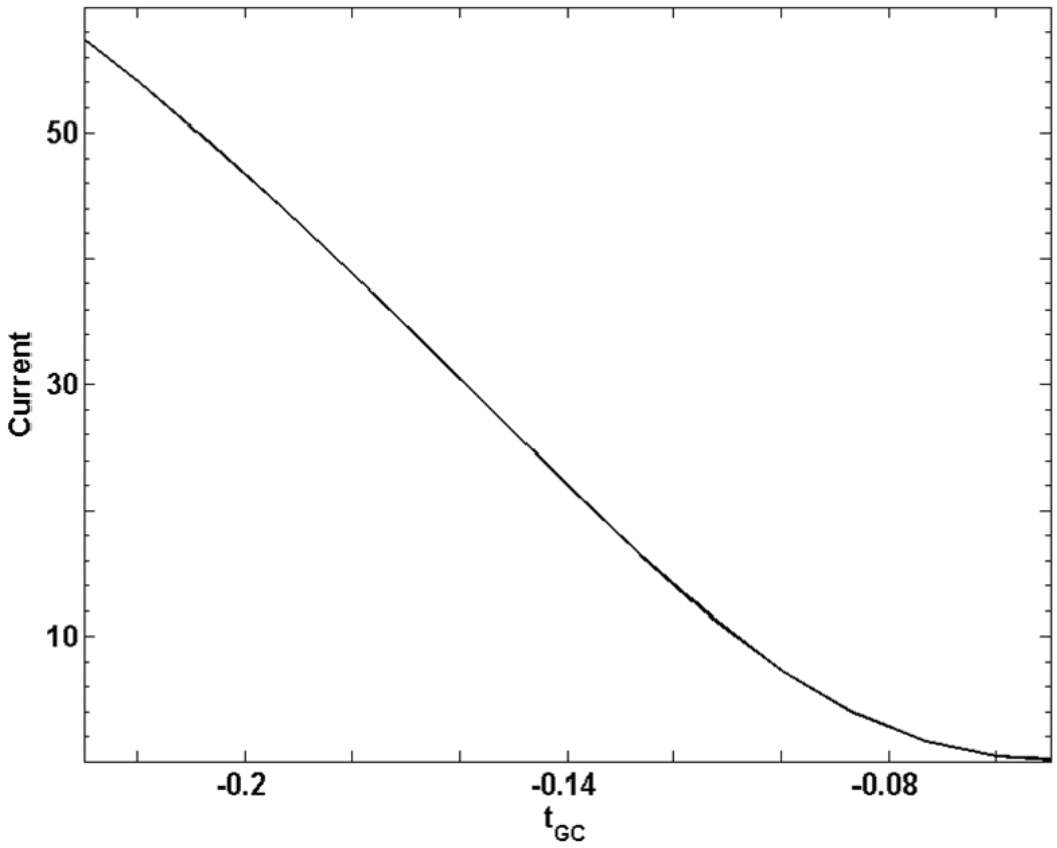


Figure 14 : Dependence of I-V characteristic on the transfer integral : as coupling increasers the current through the DNA also increases. This can be attributed to the widening of the energy bands due to increasing transfer integrals.

Figure 15 : Schematic of DNA energy band broadening due to coupling to the contact

Figure 14 shows the dependence of current through DNA on the value of the transfer integral increases between neighboring base pairs. The increase in current can be attributing to the increase in molecular orbit overlap between adjacent base pairs. On the other hand if the overlap between adjacent molecular orbital decreases one would expect to see a reduction in current.

Section 3.07 Effect of leads on IV characteristic of DNA

As shown in the Figure 15, the DNA in our discussion is connected to the two contacts at the end. The exact nature of the bonding at the contacts is still illusive (Heim, Deresmes and Vuillaume 2004). In the current investigation we have used 10 meV as the coupling parameters between the contact and the DNA. However as shown in the Figure 16, current through the DNA decreases with the increase in the coupling between DNA and the contacts (Zhu et al. 2004). As coupling between DNA and contact increases it changes the effective splitting between the energy levels at the end bases. However the energy levels of the bases in poly(G)- ploy (C) are already matched each other and has formed an extended mini band as show in Figure 15. The additional split introduced by the contact coupling changes the effective energy level match between the end base pairs and the rest of the molecule. If the additional splitting due to the contact is large then the mismatch between end base pairs and the rest of the molecule could effectively cause the net current to diminish. The Figure 16 shows the current change as a function of coupling between leads and the DNA and Figure 17 shows the change in transmission at different energies as a function of coupling between leads and the DNA. Note

that in the Figure 16 and Figure 17 the color corresponds to the current and transmission respectively. The color values are in log scale. This destroys the strep nature of the Figure 16. In fact all I-V curves show steps due to the presence of discrete energy levels. In summary the transmission is maximized if the coupling between DNA and contact is roughly equal to the transfer integral between adjacent base pairs of the DNA.

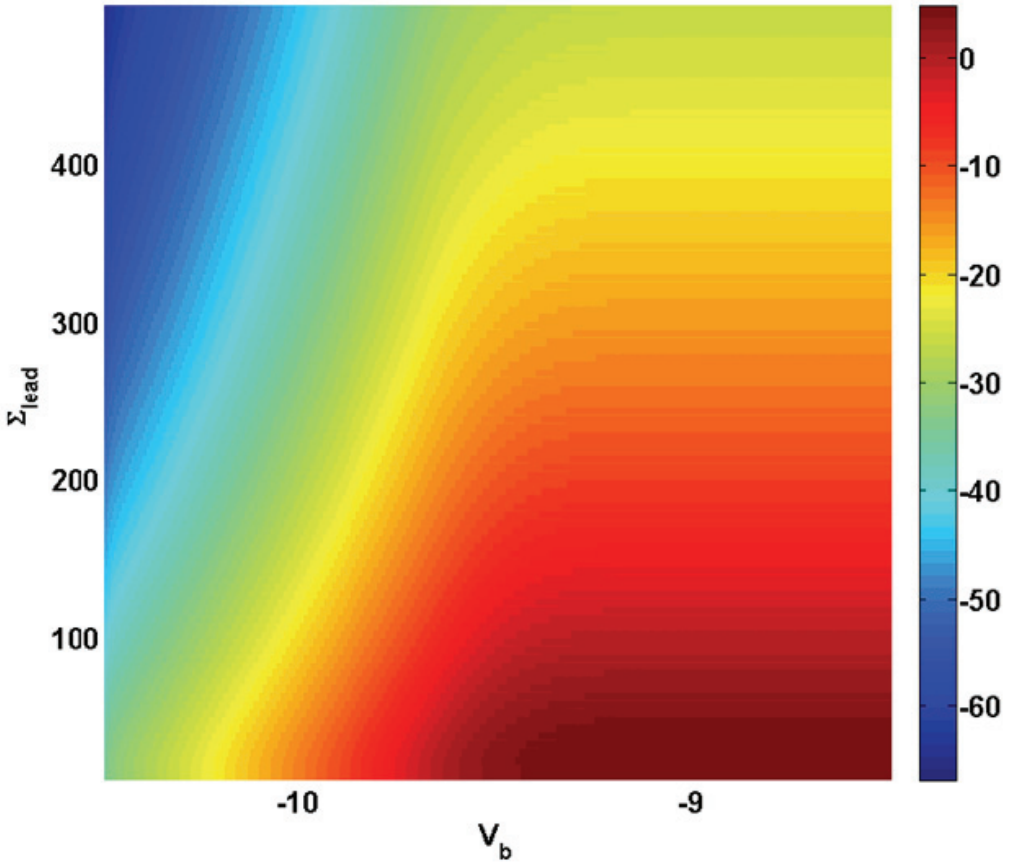


Figure 16 : Current as a function of the coupling between DNA and leads.

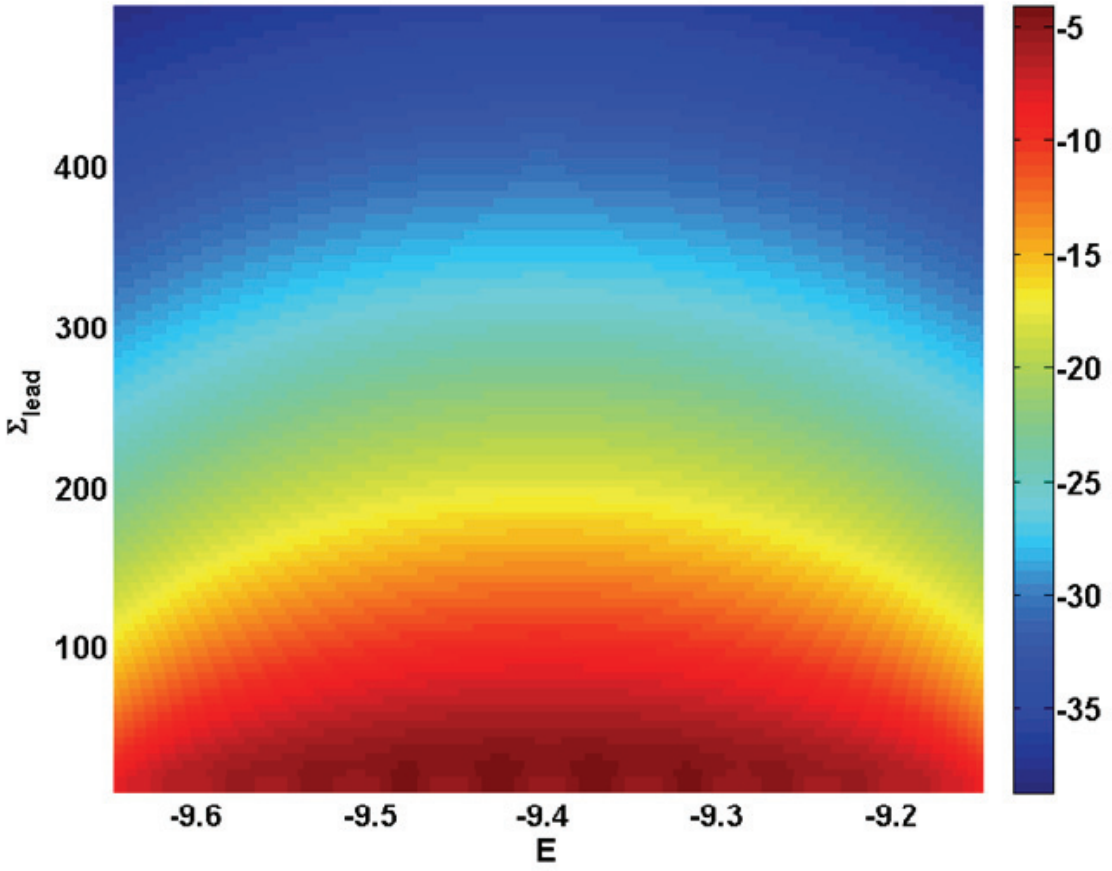


Figure 17 : Transmission as a function of coupling between DNA and leads.

Section 3.08 A mismatch base pair: Defect in the DNA

Base pairs of DNA have self assembly property. Almost all the time Cytosine (C) and Guanine (G) pairs each other forming G-C base pair whereas Adenine (A) and Thymine (T) forms A-T base pair. But on rare occasions this naturally favored ordering can be broken, result in a mismatched base pair. A mismatched base pair such as Guanine (G) and Adenine (A) or Cytosine (C) and Thymine (T) base pair could result in structural distortion in the DNA strand and hence change the effective overlap between adjacent molecular orbital. Even though this

could results in strain hence higher energy; the highly flexible helical structure of DNA permits for formation of relatively stable defects (Zhang et al. 2002, Fixe et al. 2005). The identification of such defects is mainly carried out by repair enzymes based on structural abnormalities. But the defects form by the mismatch base pair, Guanine – Adenine (G-A), resembles the least structural changes among the all other mismatched base pairs and difficult to identify by means of structural factors. One of the alternative ways that a mispair could be indentified is through the measurement of current through the DNA segment containing mispair. This mechanism is postulated as the one of the main mechanism that repairs enzymes used to detect defects in the live cells. A sender and receiver are assumed to be moving along the DNA and measure the current through it. Though it could be very slow process on a two meter long DNA in a cell, compareed to the time scale of cell division it may be sufficiently fast mechanism. In order to get an insight of this process we have investigated the I-V characteristic of DNA strands containing mismatched G-A based pairs.

First step toward this procedure was to compute the transfer integral between G-A mismatched base pair. The optimal structure for the DNA sequence without mismatched base pair is readily available in the literature. The command starting point is to use the structure of the B-DNA (Drew et al. 1981). A mispair is introduced into the optimized B-DNA structure using suitable molecular modeling program. Throughout the current work the molecular modeling and structural optimizations was performed with in CHARMM molecular modeling program. During the optimization procedure, adopted basis Newton–Raphson minimization procedure was used to find global minima of the energy surface.

The optimized structure is then used to compute the charge transfer integral between mismatched base pair and the regular Watson-Crick base pairs. As can be seen from Table 1, for some DNA

sequences the replacing of the G–C base pair by mispair increases the overlap of the $\pi$ orbital of the nearest-neighbor nucleobases, i.e. increases the intrastrand hopping integral. The magnitude of the intrastrand charge transfer integrals depends on the distance and the twist angle between nucleobases (Marra and Schär 1999). It was shown in (Marra and Schär 1999) that even small fluctuations of the twist angle by 5° can change the charge transfer integral by 0.1 eV. As a G・A mispair induces only local changes in DNA structure, it mainly changes the equilibrium twist angle. Thus the G・A mispair can result in both enhancement and suppression.

Table 2 Charge transfer integral for homogeneous poly(A)-poly(T) DNA strand with mispair

| 5'–A–X1–A–3' | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3'–T–X2–T–5' | | | | | | | |
| (X1–X2) | $t_{H_{1-2}}$ | $t_{H_{2-3}}$ | $t_{L_{1-2}}$ | $t_{L_{2-3}}$ | $t_{\perp_1}$ | $t_{\perp_2}$ | $t_{\perp_3}$ |
| A-T | 0.011 | 0.011 | -0.075 | -0.075 | 0.054 | 0.054 | 0.054 |
| G(anti).A(anti) | -0.002 | -0.055 | -0.079 | -0.117 | 0.054 | 0.131 | 0.054 |
| G(anti).A(syn) | 0.027 | 0.207 | -0.092 | −0.011 | 0.054 | -0.161 | 0.053 |

For the homogeneous chains, such as $(G–C)_3$ and $(A–T)_3$ sequences, the interstrand hopping integrals $t_\perp$are identical for all base pairs, while for sequences containing a mispair in the middle of a chain (Edirisinghe et al. 2010), the interstrand hopping integral, $t_{\perp_2}$ , corresponding to the mispair differs from $t_{\perp_1}$ and $t_{\perp_3}$. A similar tendency was observed for the intrastrand integrals $t_L$ and $t_H$ , where the presence of a mispair changes the hopping integrals. Therefore, incorporation of a mispair into the $(G–C)_3$ or $(A–T)_3$ DNA sequence leads to a change of sign and the magnitude of the interstrand and intrastrand transfer integrals, which should affect the transport characteristics of the DNA. However, in addition to alteration of the charge transfer integrals in the presence of a mispair, modification of the energetic profile within the DNA chain containing a mispair occurs, which would also significantly influence the charge transfer within DNA.

Table 3 : Charge transfer integral for homogeneous DNA strand with mispair

| 5'–G–X1–G–3' | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3'–C–X2–C–5' | | | | | | | |
| (X1–X2) | $t_{H_{1-2}}$ | $t_{H_{2-3}}$ | $t_{L_{1-2}}$ | $t_{L_{2-3}}$ | $t_{\perp_1}$ | $t_{\perp_2}$ | $t_{\perp_3}$ |
| G-C | −0.133 | −0.133 | −0.041 | −0.041 | −0.050 | −0.050 | −0.050 |
| A-T | −0.218 | −0.102 | 0.066 | 0.067 | −0.001 | −0.063 | −0.001 |
| G(anti).A(anti) | −0.213 | 0.146 | −0.072 | −0.071 | 0.024 | 0.020 | −0.025 |
| G(anti).A(syn) | 0.254 | −0.136 | −0.153 | −0.073 | 0.024 | 0.091 | −0.025 |

As explained earlier, for homogeneous poly(G)–poly(C) DNA the coupling between the nearest-neighbor base pairs results in the formation of HOMO and LUMO minibands. The widths of these minibands are determined by the intrastrand hopping integrals. Introduction of a mispair in such a homogeneous DNA results in scattering of otherwise 'freely' propagating charges. This scattering suppresses the current through the DNA. Figure 18, the current through DNA is shown as a function of the applied bias voltage for a poly(G)– poly(C) DNA without and with a mispair. The typical $I - V$ characteristic has a step-like behavior, where transitions between the steps occur when the chemical potentials of the contacts cross the HOMO or LUMO energy bands. The steps are clearly visible in Figure 18 for DNA without and with a mispair. The actual positions of the steps depend on the gate voltage, while the widths of the steps are determined by the HOMO–LUMO energy gaps. The width of the transition region between the steps is determined by the width of the HOMO and LUMO minibands, i.e. by the values of the hopping integrals, and the temperature. In what follows, we study the saturated value of the current through the DNA. In Figure 18, this saturated value corresponds, for example, to the bias voltage of −4 eV. At this bias voltage both the HOMO and LUMO states contribute to the current through the molecule. The saturated value of the current also has a weak dependence on the temperature of the system.

As already mentioned above, we consider below only two types of mispair: the G・A mispair in two of its conformations: G(anti)・A(syn) and G(anti)・A(anti) (Brown et al. 1989). Within the

tight-binding model the differences between these two conformations are the values of the hopping integrals between the mispair and the nearest-neighbor canonical base pairs. For both types of conformations, the on-site energies are the same. Figure 18 clearly illustrates that for both types of mispair the current through the DNA is suppressed with the introduction of a mispair. The stronger suppression of the current is caused by the G(anti)・A(syn) mispair. For a homogeneous DNA sequence a mispair can be considered as the defect which breaks the periodicity of the tight-binding model. Such a defect results in additional scattering and finally suppresses the current through DNA.

There are two modifications the mispair introduces into the homogeneous system: a mispair changes the on-site energy and a mispair changes locally the hopping integrals. Both of these effects result in suppression of the transport through the chain, although the transport is more sensitive to the changes in the intrastrand hopping integrals. Comparing the hopping integrals for G(anti)・A(syn) and G(anti)・A(anti) mispair (see Table 1), we can conclude that the changes in the hopping integrals for the G(anti)・A(syn) mispair is larger than for the G(anti)・A(anti) mispair. As a result suppression of current through a DNA chain is larger for a G(anti)・A(syn) mispair, which can be seen in Figure 18. A similar suppression of the charge transport through homogeneous DNA should be expected if the G・C base pair is replaced by a canonical A・T base pair. In this case we also see the suppression of the current, which is shown in Figure 18. The saturated values of the current through DNA molecules with a A・T base pair and a G(anti)・A(anti) mispair are close, which can be explained by similar values of the hopping integrals for a A・T base pair and a G(anti)・A(anti) mispair (see Table 1).

To describe the effect of a mispair on the electrical current through DNA, we introduce two characteristics. The first one is the change of the saturated value of the current through DNA at a large bias voltage, $V_b = -4$ eV.

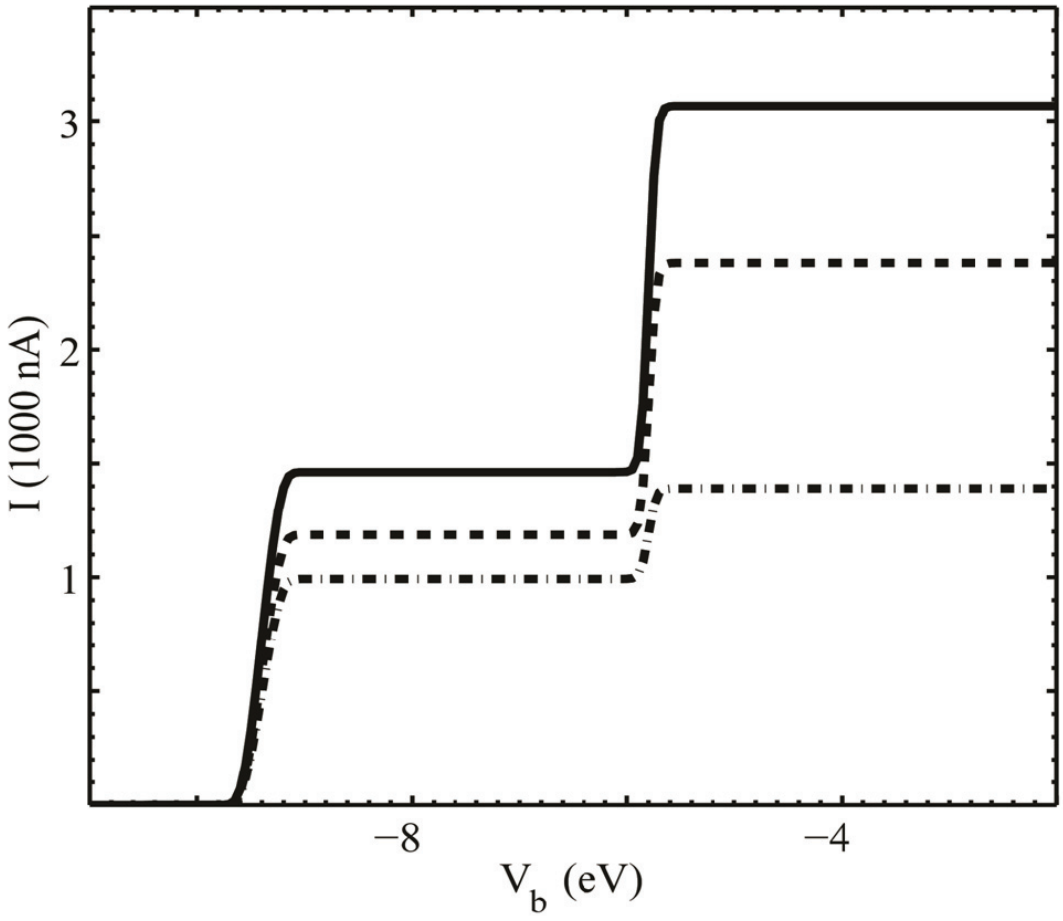


Figure 18 : I–V characteristic of homogeneous poly(G)–poly(C) DNA molecule with different types of base pairs in the middle of the molecule: regular poly(G)–poly(C) molecule without any (dashed–double dotted line); poly(G)–poly(C) molecule with G*(*anti*)*・A*(*anti*)* mispair (solid line); poly(G)–poly(C) molecule with G*(*anti*)*・A*(*syn*)* mispair (dashed–dotted line).

This change is calculated as the difference between the current through the DNA without a mispair and that when a mispair is present in the DNA: equation (26).

$$\Delta I = I_{with\ out\ mispair} - I_{With\ mis\ pair} \tag{28}$$

Another characteristic is the relative change of the current, which is defined by the following expression:

$$\delta I = \frac{\Delta I}{I} \tag{29}$$

The relative change of current is a more appropriate characteristic of the effect of a mispair, especially when the dependence on the length of the DNA is considered.

Since the mispair modifies the structure of DNA only locally, which means that the parameters of the tight-binding model are changed only near the mispair, the changes in the current due to the presence of the mispair should be suppressed with increasing length of the DNA wire.

In Figure 19 the change of the current through DNA due to the introduction of a G(anti)・A(syn) or a G(anti)・A(anti) mispair is shown as a function of the length of the DNA. For all values of the DNA length the suppression of the current is stronger for the G(anti)・A(syn) mispair.

The results shown in Figure 19 illustrate that the suppression of current through DNA with a mispair is discernible even for a long DNA with up to 90 base pairs. Although the decrease of $\Delta I$ with increasing DNA length is clearly seen in Figure 19, the current, $I$ without mispair, itself is also suppressed. As a result, the relative changes $\delta I$ of the current, shown in Figure 20, remains almost the same. For the G(anti)・A(anti) mispair the relative change is $\delta I \approx 0.22$, while for the G(anti)・A(syn) mispair it is $\delta \approx 0.55$. In both cases, $\delta I$ has a weak dependence on the length of DNA. The reason for such a weak dependence is the homogeneous sequence of DNA without a mispair. For such a sequence the corresponding wave functions have the form of propagating waves, $e^{-ikx}$, extended over the entire DNA. As a result, each of the electron wave functions becomes sensitive to the presence of a mispair and the effect of the mispair on the wave function is appreciable even at large distances. The effect of a mispair on the current through DNA

depends also on the actual position of the mispair within the molecule. In the above analysis we assumed that the mispair is placed exactly in the middle of the DNA chain. In Figure 21 the relative change, $\delta I$ , of the current through DNA is shown as a function of the position of the mispair.

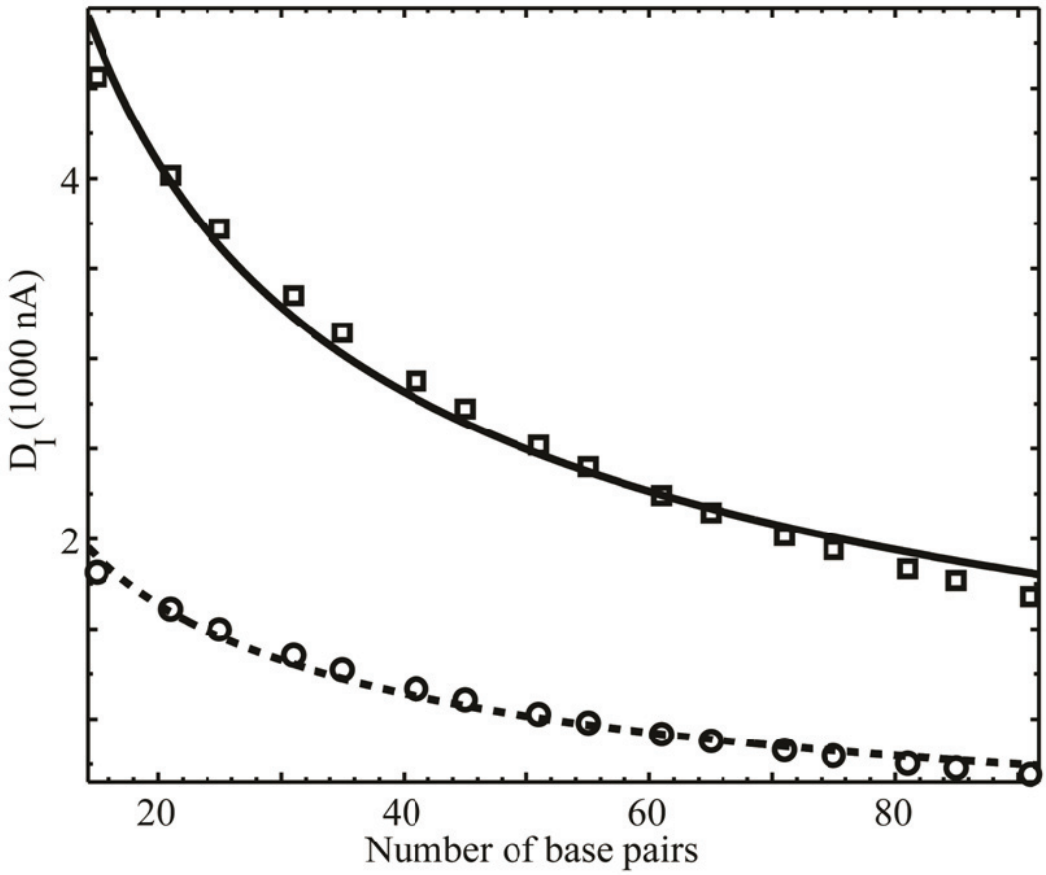


Figure 19 : The change, ΔI, of the current through DNA as a function of the length of DNA (the number of base pairs in DNA). The change of the current is defined as the difference between the current through the DNA without a mispair and that for the DNA containing a mispair: G*(*anti*)* · A*(*syn*)* (solid line) or G (anti) · A(anti) (dashed line). The mispair is placed in the middle of the homogeneous poly(G)–poly(C) DNA.

For both types of mispair the maximum changes in the current occur when the mispair is close to the middle of the molecule, although the dependence on the position of the mispair is weak. There is also some irregularity in the dependence of $\delta I$ on the position of the mispair. This irregularity is due to the extended nature of the wave functions of charge carriers in the

homogeneous DNA. For a homogeneous DNA sequence, all electron wave functions corresponding to the tight-binding model are extended and occupy the whole DNA molecule. Therefore, all electron states are sensitive to the presence of a mispair which results in strong changes, $\delta I$ , in the current through the DNA and also in the dependence of these changes on the position of the mispair.

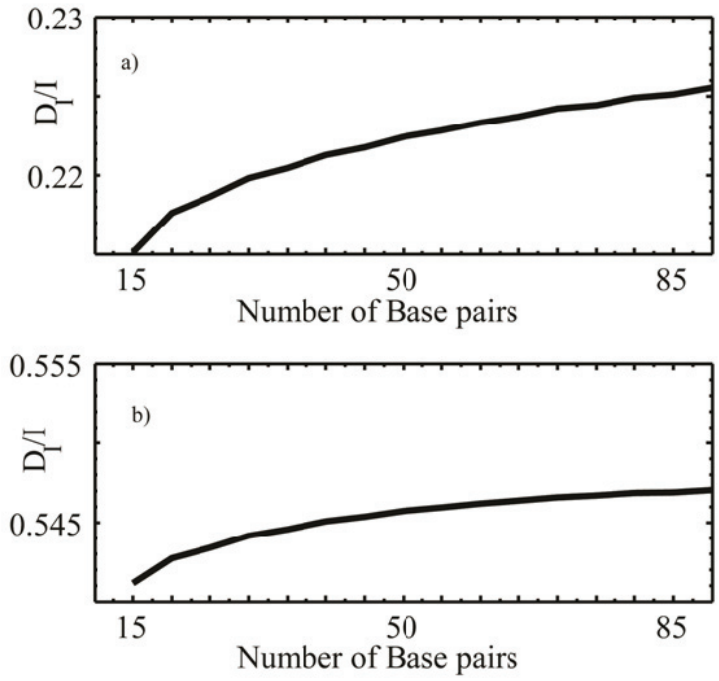


Figure 20 : The relative change of the current , $\delta I = \Delta I / I$, as a function of the length of DNA containing a (a) G(anti)・A(anti) or a (b) G(anti)・A(syn) mispair. The mispair is placed in the middle of homogeneous poly(G)–poly(C) DNA.

## Section 3.09 Inhomogeneous DNA Strands

For an inhomogeneous DNA sequence the situation is different. The electron states for such a DNA become more localized and the mispair modify only a few of them. In order to study the properties of an inhomogeneous DNA sequence, we consider a random sequence of DNA with a

G–C canonical base pair or one of the mispair, G(anti)・A(syn) or (G(anti)・A(anti)), in the middle of the molecule.

For each of such random sequences we calculate the relative change of current when the Watson–Crick G–C base pair is replaced by the mispair. The corresponding changes in the current are calculated for 500 different random realizations of DNA sequences. We present the results of our calculations in terms of the probability density function of the relative change of the current through DNA, $P(\delta I)$.

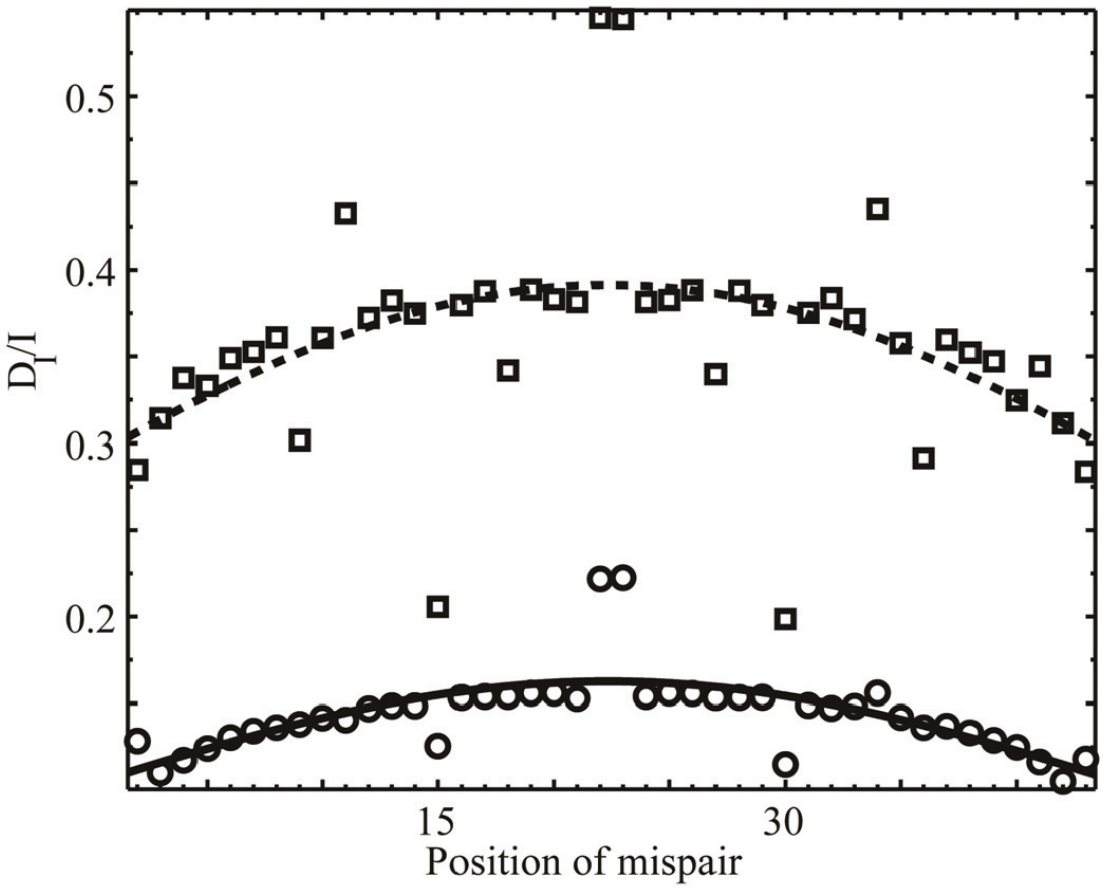


Figure 21 : The relative change , $\delta I = \Delta I / I$ , of the current through a homogeneous poly(G)–poly(C) DNA versus the position of the mispair: G*(*anti*)*・A*(*syn*)* (dashed line) and G*(*anti*)*・A*(*anti*)* (solid line). The DNA consists of 45 base pairs.

In Figure 22 the probability density function is shown for the G(anti)・A(syn) and G(anti)・A(anti) mispair and for DNA molecules consisting of 10 and 15 base pairs. For both types of mispair the probability distribution function has a sharp maximum at $\delta I$ close to 1. The value of $\delta I$ =1 means that the current through DNA with a mispair is zero, i.e. it is much smaller than the current through DNA without a mispair. Therefore, for the corresponding DNA sequence, introduction of a mispair strongly suppresses the current through DNA ($\delta I \approx 1$).

This strong suppression is more pronounced in the case of the G(anti)・A(syn) mispair, for which the corresponding peak of the probability density function is narrow with a maximum at $\delta I \approx 0.9$ and a width of ≈0.2. For the G(anti)・A(anti) mispair the peak is broader with a maximum at $\delta I \approx 0.8$ and a width of ≈0.4. The stronger suppression of the current for DNA with a G(anti)・A(syn) mispair can be related to the larger changes in the intrastrand hopping integrals due to the presence of a mispair compared to DNA with a G(anti)・A(anti) mispair, see the values of $t_H(1)$ and $t_L$ in Table 1. Therefore the G*(*anti*)*・A*(*syn*)* mispair results in stronger modification of the tight-binding parameters of the DNA chain.

From the distribution functions shown in Figure 22 we can infer that for many (about 78%) generic (random) DNA sequences, introduction of a G(anti)・A(anti) mispair results in a suppression of the current through DNA, i.e. for such DNA molecules $\delta I > 0$. There is also a very broad region with negative values of $\delta I$ , which corresponds to an enhancement of the current through DNA. For a short DNA (10 base pairs in Figure 22 (d)) there is also a local peak at $\delta I = 0$, which disappears for longer DNA (Figure 22 (b)). Here $\delta I = 0$ means that the current

through DNA does not change with introduction of a mispair. A small peak at $\delta \approx -6$ is due to the finite size effect and disappears at larger lengths of DNA.

For the G(anti)・A(syn) mispair we have a different behavior (see Figure 22 (a) and (c)). Similar to the case of G(anti)・A(anti), there is a large number of different DNA sequences for which the current through DNA is suppressed with the introduction of a G(anti)・A(syn) mispair ($\delta I > 0$), but this number depends on the length of DNA. For example, for a DNA with 10 base pairs there are 78% of generic DNA sequences for which the current through DNA is suppressed.

For a DNA with 15 base pairs it is 66%. Hence with increasing length of DNA the number of different DNA sequences with $\delta I > 0$ decreases. For a G*(*anti*)*・A*(*syn*)* mispair there is also a pronounced peak near $\delta I = 0$. This peak disappears when the length of the DNA is increased. For negative values of $\delta I$ we have an enhancement of the current through the DNA.

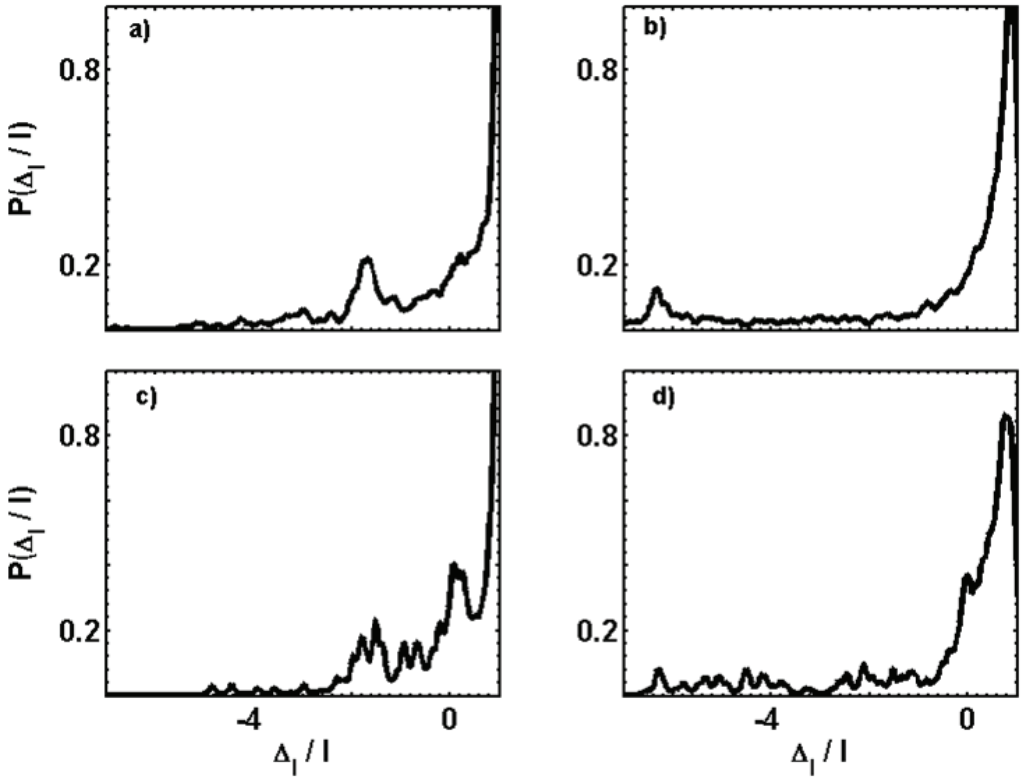


Figure 22 : Probability distribution of current with mispair: a) and b) probability distribution for DNA sequence of 15 base pairs with G(anti)-A(anti) and G(anti)-A(syn) mispair. c) & d) for DNA sequence with 10 base pairs.

Table 4 : Current change for different inhomogeneous sequences.

| DNA sequence | G(anti)·A(syn) | G(anti)·A(anti) |
|---|---|---|
| A A G G G A G G A | -1.5039 | -5.7986 |
| A A A G G G G G G | -1.5071 | -2.6232 |
| G A G G G A A G A | 0.0781 | 0.7405 |
| G G A G G G A G A | 0.0750 | 0.1506 |
| G G A A G G G G A | 0.8987 | 0.8691 |
| G A A A G A G A A | 0.9028 | 0.7104 |
| A G G A G A A A A | 0.2788 | 0.7377 |
| G A A G G G A A G | 0.9977 | 0.74 |
| G G G G G A G A G | 0.3575 | -0.0997 |
| G A A G G A A G G | -0.3820 | 0.0420 |

Although a mispair can be considered as a defect in a DNA chain, the effect of a mispair on the charge transport through the molecule can be different for different DNA sequences. If the current through DNA without a mispair is large, i.e. there are well-connected transport paths through the DNA molecule, then the mispair in such a system become a real defect which suppresses the transport. In this case we observe suppression of the current through DNA. If the current through DNA without a mispair is originally small, introduction of a mispair in such a system can open additional transport channels, which can increase the current through the molecule.

Although the enhancement of the current through DNA exists for both types of mispair, the more pronounced enhancement is visible for a G(anti)・A(syn) mispair. This can be related to stronger modifications of the intrastrand hopping integrals by a G(anti)・A(syn) mispair. In table 3 a few random DNA sequences corresponding to the peaks in Figure 22 are shown with the corresponding values of $\delta I$ for different conformations of the G・A mispair. There is some correlation between the values of $\delta I$ for the G(anti)・A(syn) and G(anti)・A(anti) mispair, that is, $\delta I$ is either positive or negative for both types of mispair.

This confirms the general tendency, that enhancement or suppression of the current is determined by the DNA sequence. That is, if the current through DNA without a mispair is small then we should expect enhancement of the transport, and if the initial current is large then the mispair suppresses the transport. Such a tendency is valid for both conformations of G・A mispair.

## Section 3.10 Effect of Temperature on I-V Characteristic of DNA

In this section we will investigate the temperature effects on the I-V characteristic of the DNA. Loosely bonded DNA bases are subject to large oscillations. Among various oscillations present in the DNA, twist motion is the most important in terms of charge transport properties. Twist motion changes the twist angle between adjacent base pairs from its equilibrium value of $36^0$. Marra and Schär (Marra and Schär 1999) has showed that even $5^0$ change in twist angle can leads to significant changes in transfer integrals as much as 0.1eV. Such large, yet local, changes in transfer integrals could leads to very different conductance properties through the DNA.

In this section we take advantage of the data already published by Kubar□ et al. (Kubar□ et al. 2008). In this article authors formulate angle dependence on transfer integrals using first principle computations. However angular dependence on charge transfer integral for GA mismatched base pair is unknown. In the present work we have assumed a liner dependence of charge transfer integrals for small angles.

Steady state current was calculated using Equation (26). However in the present problem transfer matrix elements are position dependent. Finite temperature effects were added by means of random, temperature dependent distribution of the angle. The most important vibration amount all possible motions in view of charge transport properties is twist motion between two adjacent base pairs. Twist motion in turn changes the relative overlap between two adjacent base pairs alternating the charge transfer integral between them. We have introduced the twist angle perturbations by introducing random deviations from the equilibrium twist angle. Each initial random angles were assigned form the normal distribution with zero mean and standard deviation $\langle R_n(t)R_{n'}(t')\rangle = 2k_BT\zeta I\delta_{nn'}$. Where $k_B$ is the Boltzmann constant, I is the moment of inertia of the DNA base pair, $\zeta$ is the damping constant. T is the temperature.

The additional energy gained by the twist motion is expressed by the second term in the Equation 6. We have performed a ensemble average using a partition function as;

$$I = \frac{\sum_n I_n e^{-E/K_B T}}{\sum_n e^{-E/K_B T}} \tag{30}$$

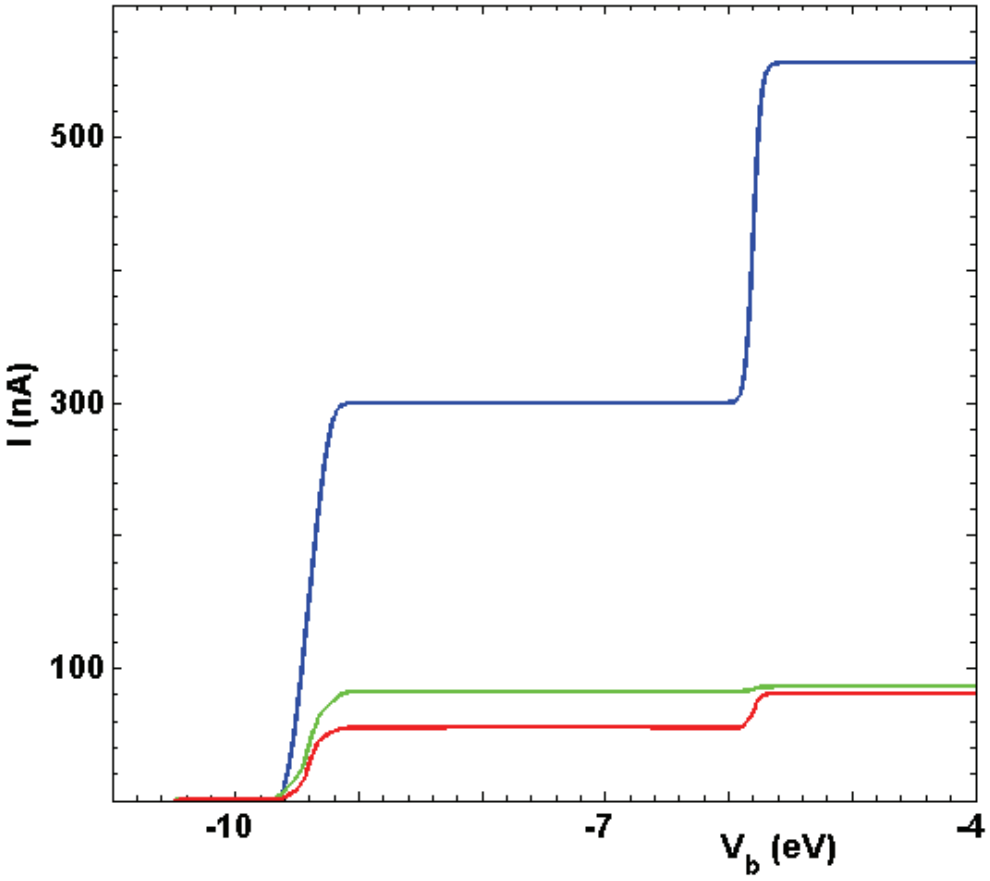


Figure 23 : effect of the temperature: Homogeneous poly(G)-poly(C) DNA strand Blue , green and red curves represents 0K,5K and 300K respectively.

Typical IV characteristic for the homogenous DNA stand at three different temperatures is shown in Figure 23. In the Figure 23, the blue curve represents the current at 0K whereas green and red curves correspond to the current at 5K and 300K respectively. As discussed in the previous sections, homogeneous DNA strand exhibits good conductance due to possibility of having continuous mini bands. However structural fluctuations introduced by temperature can

change the transfer integral from its equilibrium value. As mentioned in the previous sections, transfer integral determines the width of the minibands.

As a result, local change in transfer integral could adversely affect the formation of continuous minibands across the DNA. The decrease in current in the Figure 23, with increase in temperature, could be attributed to the loss of continuous minibands.

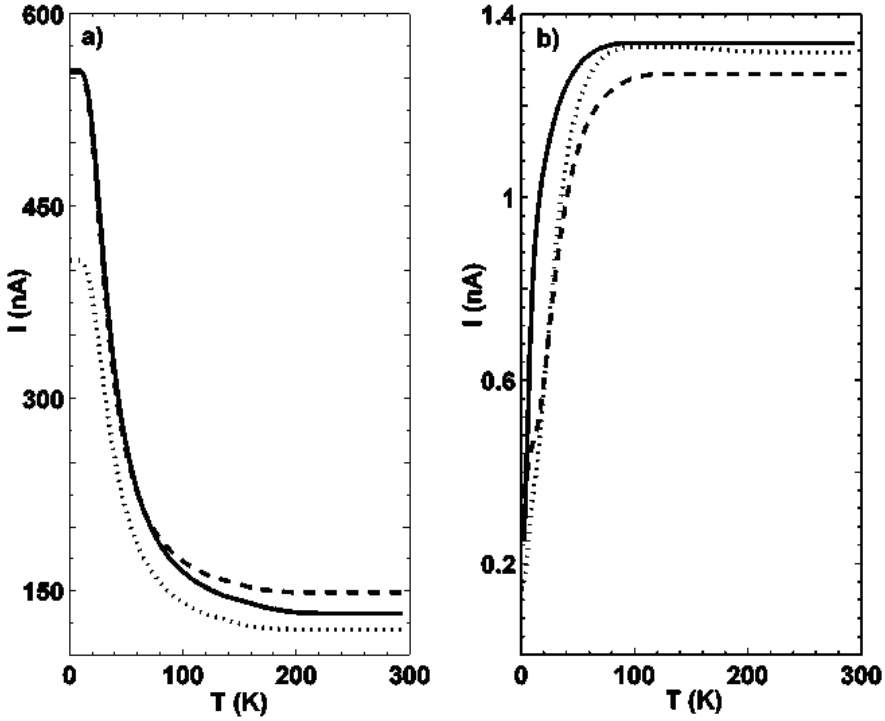


Figure 24: Temperature dependence of the Current: a) Homogenous DNA strands b) Inhomogeneous DNA strand solid curve is for homogenious DNA strand. Dotted curve is for G(anti).A(syn) mispair whereas dash curve is for G(anti).A(anti) mispair

However in the case of inhomogeneous DNA, current is inherently small. This is attributed to the mismatch between different energy levels. Transfer integrals determine the energy level splitting as two bases brought closer. Higher transfer integral results in higher energy level splitting. As mentioned above changes made in to the twist angle by thermal fluctuations can change the

magnitude of the transfer integral. This in turn can change the width of the energy band locally. Perhaps this can leads to increase of the formations of extended energy band facilitating larger current. In Figure 24 b) value of saturated current increase with rise of the temperature. It should be noted that the relative change in current is a function of particular DNA sequence. For example, in Figure 24 a) shows the saturation current through homogenous DNA which decreases with increase in temperature. This results from the destruction of the continuous minibands due to transfer integral fluctuations. Depending on the specific energy profile, temperature can induce reduction in current or increase in current.

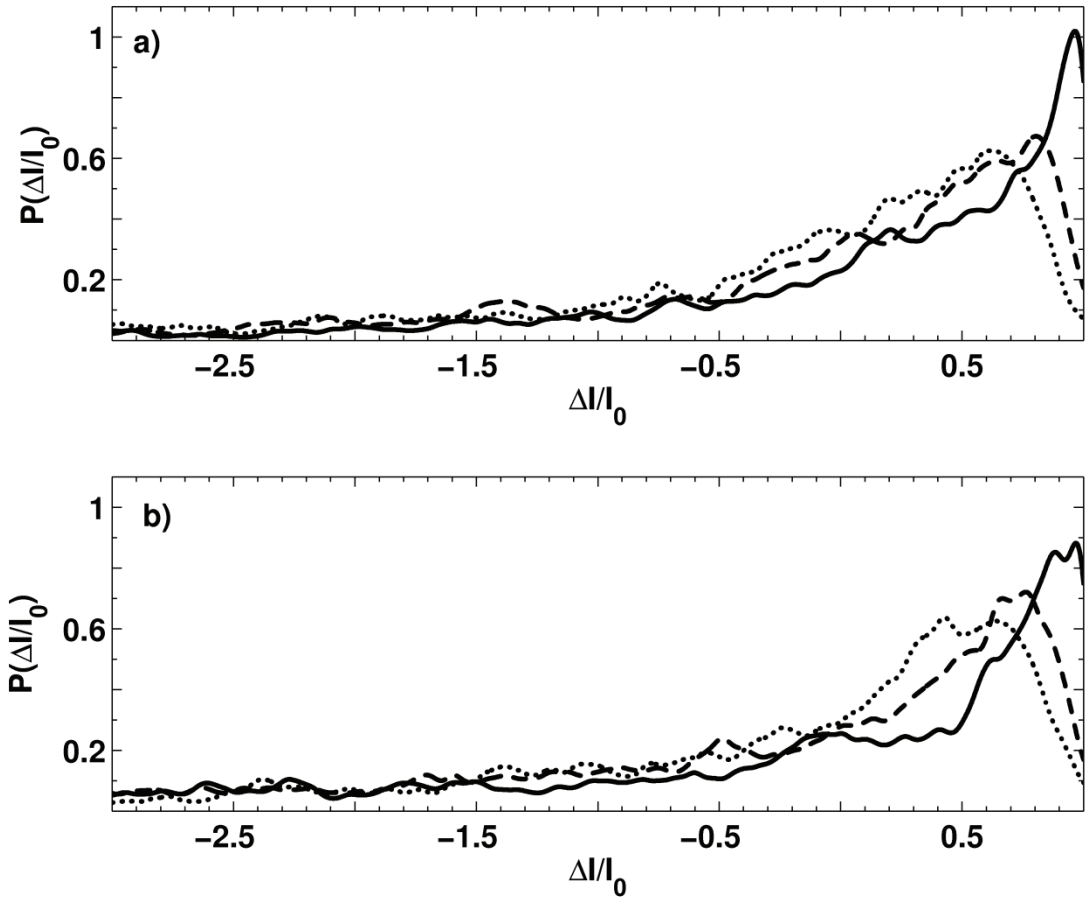


Figure 25: Distribution function for inhomogeneous DNA strands with mismatched base pairs.

Figure 25 illustrates the temperature dependence of the distribution function for the change in current (ΔI) Equation (29). The change (ΔI) is measured with respect to the DNA strand without

mispair. It should be noted that Figure 22 corresponds to 0 K situations. In the Figure 25 the solid curves represents the current at 5K dash curves represent and dotted curves represent current at 25K and 300K, respectively. As one can see from the figure temperature suppresses the effects due to the mispair.

# Chapter 4. Charge transport between Donor-Acceptor

## Section 4.01 Structural Dynamics and Charge Transport

As mentioned in the introductory chapters, the soft skeleton of the DNA permits anomalous vibration amplitudes. A charge transport through DNA can be significantly altered by these abnormal structural fluctuations. It should be noted although all atom quantum mechanics molecular mechanic (QM/MM) study could provide more detailed description into the charge transport properties through DNA, the computational complexity and computational cost hinders the utilizations of QM/MM techniques on DNA. The absence of periodic structure requires using large DNA segments, which limits the use of all atom computations. Moreover DNA is highly subject to external perturbations such as solvation effects and the action of external ions. These demands for use of significantly large water box in any MM computations. However mean field continuum approaches will also provide some valuable depiction. Within limitations of mean field approach, in order to investigate the influence of lattice vibrations on charge transport we have performed semi classical- Quantum mechanical simulations on coarse grained DNA model using the system of equations described in the Section 2.01(c) . Since realistic systems are subject to thermal fluctuations due to temperature, temperature effects are included into the system of equations by means of random force. This makes each process a stochastic process. The random force has the following properties,

$$\langle f_n(t) \rangle = 0, \tag{30}$$

$$\langle f_n(t) f_{n'}(t') \rangle = 2k_B T \gamma m \delta_{nn'} \delta(t - t') . \tag{31}$$

Where $k_B$ is the Boltzmann constant, T is the temperature, m is the mass of subunit (base pair) and γ is the damping constant.

The initial velocities used in the simulations were found from the Maxwell-Boltzmann distribution. Statistically, at longer time scale this procedure mimics the temperature effects. In order to realistically represent the system, we have included a dispersion term into the Equation (7) in the form of $-m\gamma\frac{dy_n}{dt}$ . This term represents the energy loss due to the interaction with thermal bath. Charges were captured at the acceptor site by adding a decay term $-\frac{i\hbar}{\tau}$ into the electron Hamiltonian (11). Finally coupling between charge Hamiltonian and skeleton Lagrange (hereafter will be called lattice) was introduce by adding a Hostiln type term $\chi y_n$ (Kalosakas, Aubry and Tsironis 1998, Maniadis et al. 2003, Komineas, Kalosakas and Bishop 2002), which signifies the interaction with charge with optical lattice, into the Equations (11) and (7). $\chi$ represents the charge-phonon coupling constant. This implies the onsite energy of the tight binding model linearly depends on the displacement of the site *n* as $\varepsilon_n = \varepsilon_0 + \chi y_n$ The angular dependence of charge transfer integral should be found from the first principle computations (Edirisinghe et al. 2010, Song, Elstner and Cuniberti 2008) as discussed in the previous section. However in the present treatment we model the twist angle dependent of the transfer integral by using Slater-Koster modeling (Slater and Koster 1954). In what follows we assume that each base pair posseses resultant p like orbital which contribute to the formation of π orbitals. In the current model we only consider twist motion and in plane vibration, thus only $p_z$ orbital contributes to form molecular orbital. In other words there is no mixing of the $p_z$ with $p_x$ $p_y$ orbitals which lies in the plane of vibration. In this formulation the transfer integrals can be approximate as

$$V_{ppX} = \eta_{ppX}\frac{\hbar^2}{md^2}e^{-d/R_c}, \tag{32}$$

$$V = sin^2\phi V_{pp\sigma} + cos^2\phi V_{pp\pi} = \frac{\hbar^2 e^{-d/R_c}}{md^2}\left[\left(\eta_{pp\sigma} + |\eta_{pp\pi}|\right)\frac{z^2}{l^2+z^2} - |\eta_{pp\pi}|\right]. \tag{33}$$

Finally, with all modifications mentioned above, system equation can be written as;

$$\ddot{\phi}_n = K_{\phi\phi}(\phi_{n+1} + \phi_{n-1} - 2\phi_n) - G(\phi_{n+2} + \phi_{n-2} - 4\phi_{n+1} - 4\phi_{n-1} + 6\phi_n) + \frac{K_{y\phi}}{2}(y_{n+1} - y_{n-1})$$
$$-\frac{2}{R_0}\dot{y}_n\dot{\phi}_n - \frac{2}{R_0}y_n\ddot{\phi}_n - \frac{2}{R_0^2}y_n\dot{y}_n\dot{\phi}_n - \frac{1}{R_0^2}y_n^2\ddot{\phi}_n - \frac{dV(\phi_n,\phi_{n-1})}{d\phi_n} + f(t) \tag{34}$$

$$\ddot{y}_n = \left(1 + \frac{y_n}{R_0}\right)\frac{1}{R_0}\dot{\phi}_n^2 - \left(y_n - \frac{3}{2}y_n^2 + \frac{7}{6}y_n^3\right) - K_{yy}(y_{n+1} + y_{n-1} - 2y_n) - \frac{K_{y\phi}}{2}(\phi_{n+1} - \phi_{n-1})$$
$$-m\gamma\frac{dy_n}{dt} + \chi y_n|\psi_n^2| + f(t) \tag{35}$$

$$i\hbar\dot{\Psi}_n = -V(\phi_n,\phi_{n-1})(\Psi_{n-1} + \Psi_{n+1}) - \varepsilon_n\Psi_n - \frac{i\hbar}{\tau}\Psi_n - \chi\Psi_n y_n \tag{36}$$

Here the first two equations (34), (35) describe semi classical dynamics of the DNA, while the last equation characterizes the quantum transport of the charge within DNA molecule. Below we solve the system of equations (34)-(36) with the random force (30)-(31) numerically.

(a) How does a charge change the DNA

Time evolution of the solution of the system of equations (34)-(36) is illustrated in the Figure 26 to Figure 28. Figure 26 represents the time evolution of the twist angle whereas Figure 27 and Figure 28 represent the time evolution of the electron density and the transverse stretching between the base pairs, respectively. Entire system is initially relaxed about 300ps

and an electron is then added to the system. As shown in Figure 26 to Figure 28 the newly introduced electron introduces changes in the structural dynamics of the system. For example, the Figure 26 shows the change in the twist angle. As seen on the Figure 26 the additional energy introduced by the electron is relaxed across the flexible DNA structure. In order to accurately represent the structural dynamics of DNA a sufficiently long DNA strand is needed. We have found that a DNA strand with at least 512 base pairs are needed to minimize the edge effect in the computations. Figure 27 illustrate the evolution of the charge density. As seen in the Figure a hole is introduced into the system at around 300ps. The magnitude of the charge density is represented by the color. It should be noted that the scale of the color axis is in log scale. The donor and acceptor are separated by four base pairs. As seen in the Figure 27 small fraction of the charge density leaks out across the DNA even though significant fraction is trapped at the GC donor site. Figure 28 illustrate the dynamics of base pair stretching. Similar to the Figure 26 the introduction of an electron introduces structural changes into the system. These structural changes then propagate along the DNA change.

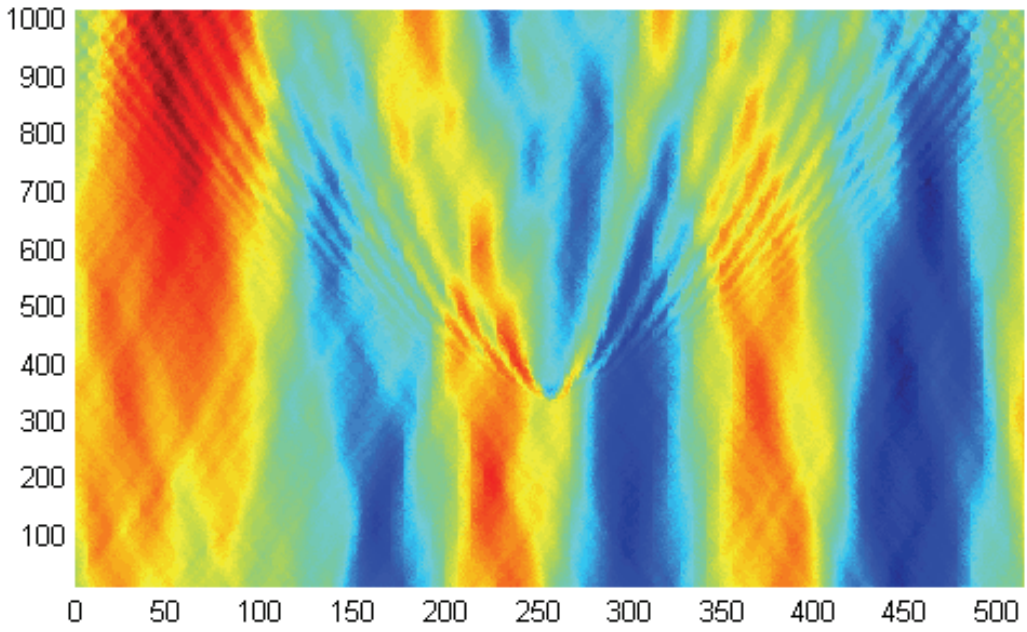


Figure 26 : the change in the twist angle with the introduction of an electron into the system. Initially system is at thermal equilibrium. As seen in the figure during the time0-300 ps DNA posses slow vibrations. At 300ps an electron is introduced into the system. The excess energy introduced by the electron then relaxes access the DNA slowly.

(b) Donor-Acceptor System

The system we study is illustration in Figure 29. The energy levels between A-T and G-C base pairs are shifted by about 0.4 eV. For example the HOMO energy level of G-A WC base pair is around 9.4 eV, whereas the HOMO energy level of the A-T WC base pair is around 9.7 eV.

Table **5** lists onsite energies for different WC bases. Within the system shown in the Figure 29 a hole transport is usually considered.

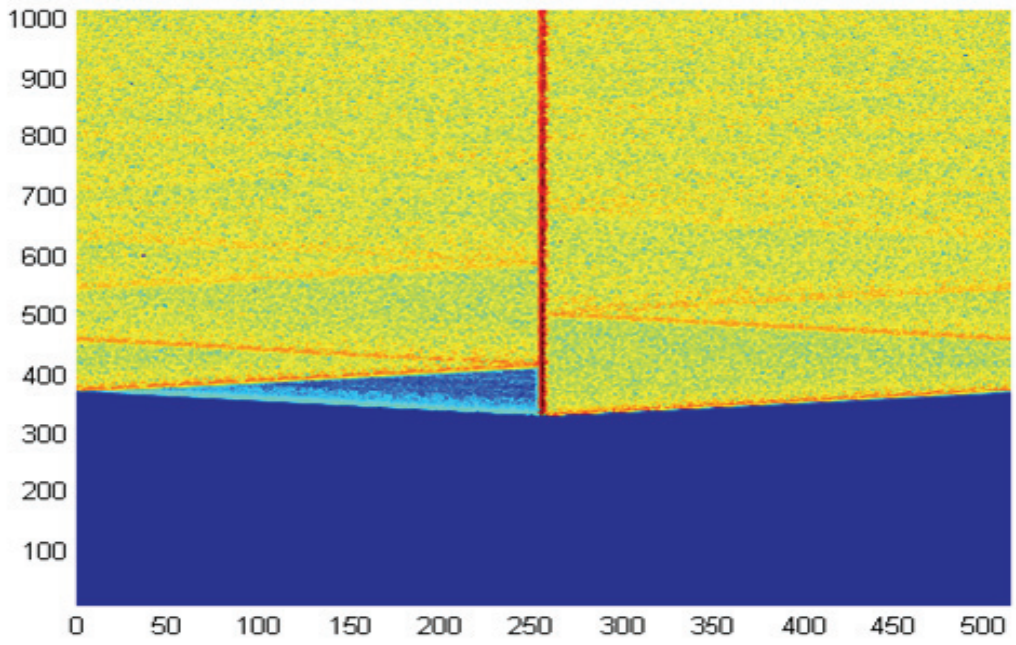


Figure 27 : evolution of the electron density with time. The color scale is in log scale. As seen from the figure even with 512 base pair fast moving electron has reflection at the boundary. But the density is comparatively small and we have safely omitted it from our consideration.

Normally, due to special energy profile of the HOMO energy levels the G-C base pair plays a role of a trap for a hole. Then the hole is added to one of the G-C base pair, the donor site, and the transfer of the hole to the other G-C base pair, the acceptor site, is studied. Our simulation consists of two G-C WC base pairs separated by variable length A-T bridges. Such system can be treated as a two quantum traps separated by a distance which is equal to the number of base pairs times 3.4 A.

(c) Can DNA trap charge?

In the present section we study the escape rate of a charge initially trapped at one of the G-C WC base pairs as a function of system parameters. The charge trapped at a GC site in a DNA differs from a pure quantum mechanical two quantum trap system due to the structural

fluctuations of the DNA skeleton. The relatively significant coupling between charge and the lattice vibration makes the present system a dissipative quantum mechanical system (Konno et al. 2007) . The energy of the electron can be transform into the lattice energy and vice versa. However the depth of the quantum trap is higher than the transfer integrals along the DNA chain. Hence we speculate the possibility of Anderson localization (Anderson 1958) in the DNA strands.

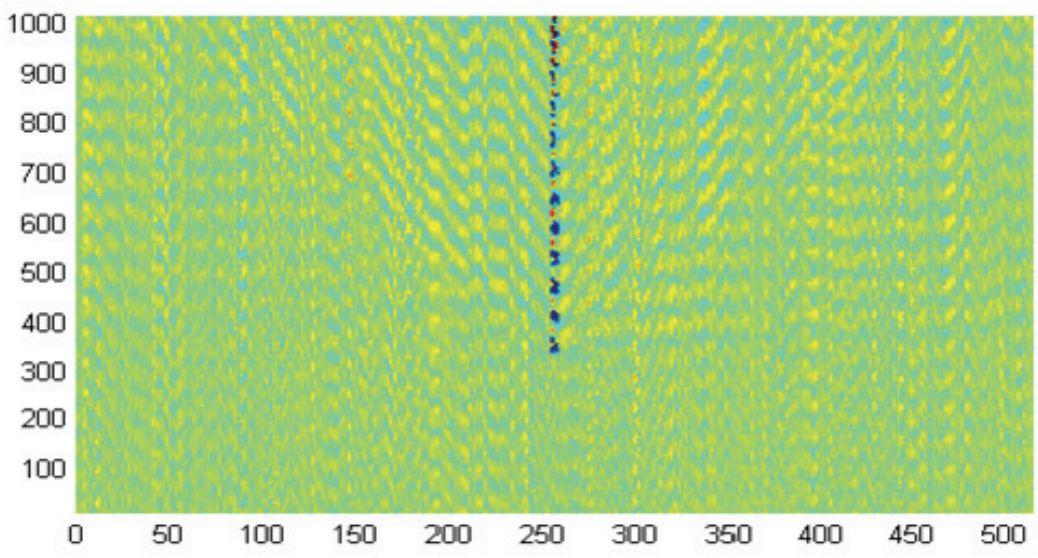


Figure 28 : the y displacement as a function of time. As shown in the figure a polaron is formed around the electron relatively soon after the electron is introduce to the system.

Table 5 : Onsite energy for isolated DNA WC base pairs.

| | G | C | A | T |
|---|---|---|---|---|
| HOMO | -9.4 | -10.27 | -9.79 | -10.46 |
| LUMO | -5.37 | -5.9 | -5.85 | -6.56 |

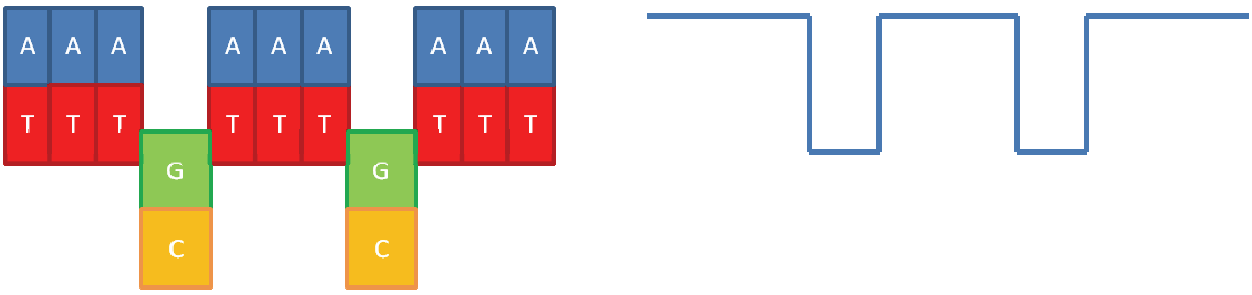


Figure 29 : A schematic representation of the energy profile in the short DNA. A) Ionization potential for G-C base pairs is differ from A-T base pairs as much as 0.4 eV. A Hole easily can localized on the G-C cite. b) Such system can be treated as two quantum trapsystems. But due to structural fluctuation system is not pure quantum system, rather a dissipative quantum system.

However due to the flexible nature of the DNA strand there exists a possibility of long range correlation between base pair fluctuations along the DNA strand. Such fluctuation inturn can introduce a long range correlation of the onsite energy disorder, facilitating electron delocalization (de Moura and Lyra 1998, Santos et al. 2006) . The DNA tight binding Hamiltonian for a charge is defined by two parameters: onsite energy and the charge transfer integral between the neighboring sites. The escape rate of trapped charge at one GC site (donor) depends on the depth of the quantum trap. For example, higher quantum trap makes it difficult for a charge to escape. However due to small dimension of the system, quantum properties are also plays an important part. If two quantum traps, i.e. G-C base pairs, are sufficiently close to each other, the charge can tunnel through the inter-trap barrier instead of completely escaping to the bridge. On the other hand DNA structural fluctuation introduces additional changes to the energy profile of the system. Depending upon the dynamics of the energy profile of the system, escape rate can increase or decrease. In this study we try to understand the importance of the structural fluctuations in order to gain realistic account on charge transport properties through DNA.

As shown in the Figure 26 - Figure 28 introduction of an electron into the system changes the system dynamics. As seen in the Figure 26 with introduction of an electron, the local distortion caused by the charge lattice interaction starts to move across the DNA in a form of a traveling wave. If DNA is not long enough these waves can reflect back or enter at the other side of the DNA, as we utilize a periodic boundary conditions. With time, relatively high amplitude resonance can develop, due to the formation of standing waves. Such high resonance could change the energy profile and significantly alter the charge escape rate. In order to study this effect we have varied the length of the DNA. As seen in Figure 31 the DNA strand with at least 512 base pairs are needed to accurately describe charge transport properties throughout the entire coupling regime that we have studied. In the Figure 31 escape time for a charge trapped in a donor GC site, which is separated from an acceptor GC site by four bases pairs apart was studied. It can be seen in the Figure 31 that for DNA length larger than 512 base pairs the escape time become independent of system dimensions. It should be noted that for smaller system size the charge escapes very rapidly see Figure 31. This is due to large structural fluctuations formed by formation on standing waves, which help in fast escape.

The results described hereafter are for a DNA strand with 512 base pairs. Each base pair is described by six variables: The complex wave function, displacement of the base pair from it equilibrium position, displacement of the twist angle from its equilibrium value, linear velocity of the bases along the transverse direction, and the angular velocity of the base pairs along the twist motion. The system of equations describing the system now consists of 3072 variables. We have solved this system using stochastic Runge-Kutta fourth order algorithms. The detailed description of the algorithm and the programming are given in the appendixes. As it was

mentioned above the effect of the temperature is taken into the account by introducing a random force.

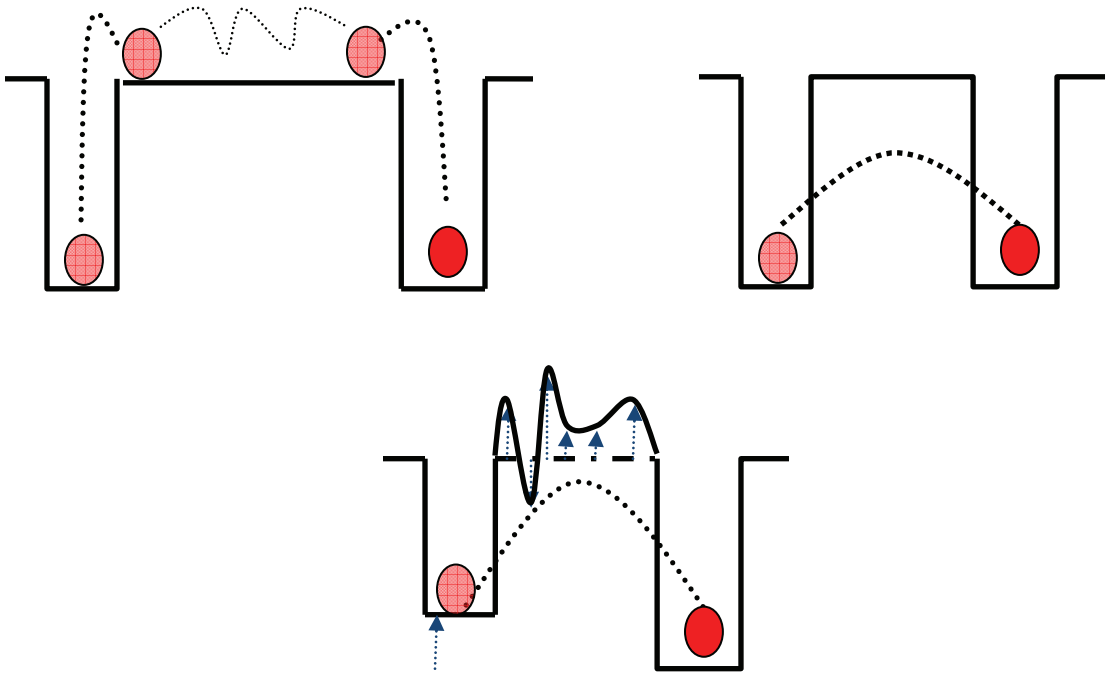

Figure 30 : Charge transport in quantum systems: a) A charge (hole) located at a quantum trap needs to gain enough energy in order to overcome the energy barrier and jump into the nearest site. b) However if two quantum traps are sufficiently close to each other then charge can take a shortcut. It can tunnel through the barrier and reach the other quantum trap. This phenomenon has quantum mechanical nature. c) the structural fluctuation introduces an additional disorder into the energy profile of the system. Depending on the new energy profile the charge escape rate can be either enhanced or suppressed.

(d) Modeling temperature fluctuations

We have performed 60 different realizations for each data point. For example when we studied charge lattice interaction for given charge lattice coupling parameter, say, charge lattice coupling constant is 1.2, we have perform 60 different realizations for that particular set of values. However we have pre-prepared the initial condition of the state variable, such as velocities and

displacements, at each temperature, by performing separate sets of simulations where system of equations given in Equation (34)-(35) was advanced in time with time step of $10^{-4}$ ps without any hole, i.e. without Equation (36).

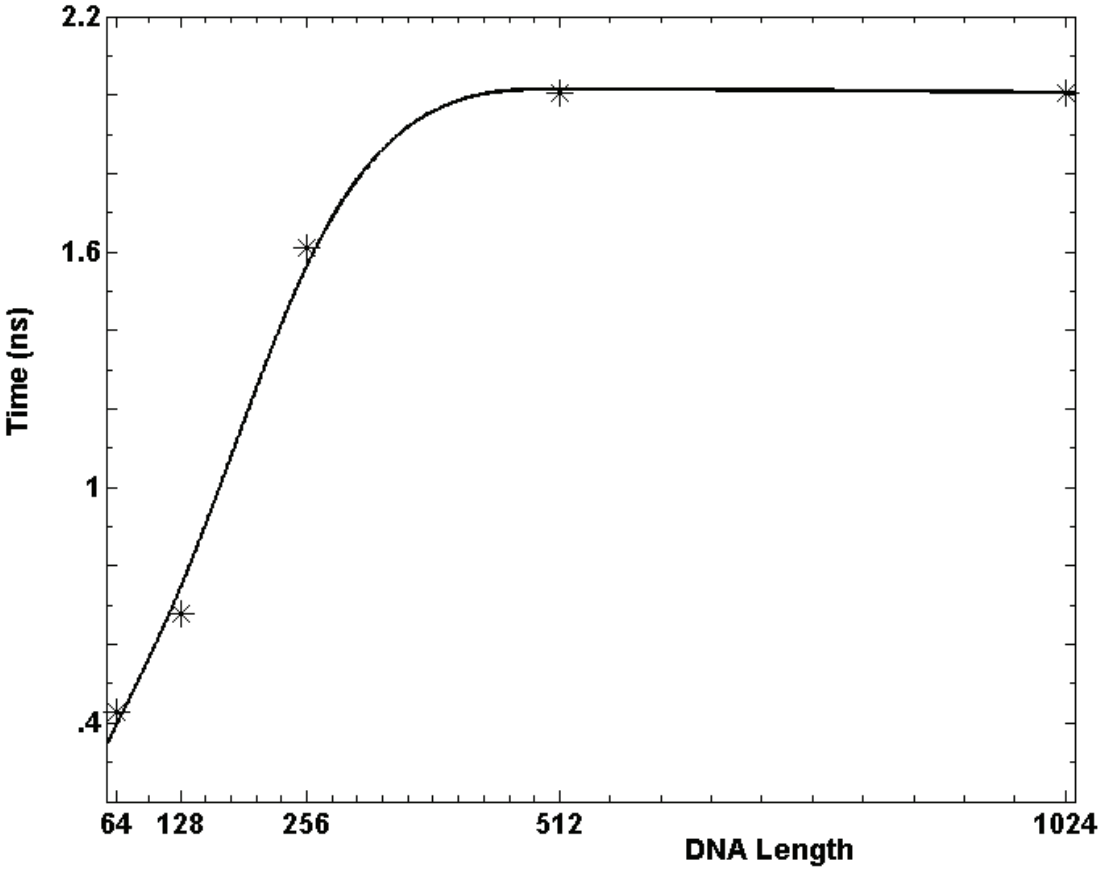


Figure 31 : the finite size effect on the escape time. The lattice charge coupling constant is 1.2 $eVA^{-1.}$ Temperature is 160 K, and the separation between donor and acceptor site is 4 base pairs.

For this pre-run velocities were selected from Maxwell-Boltzmann distribution where as displacements were selected from normal distribution. Then we slowly heat up the system to different temperatures. During this heating process small damping, the γ value, was used as it helps the system to reach equilibrium more quickly. Each simulation last about 400ps in simulation time. For example at 10K we allow system to equilibrate for 400ps. Then we save the

final values of the state variables for future use. At this point temperature of the system is rise to the next value, say, 20K and allows simulation to run another 400ps. By this way we generate 60 different initial sets of state variables at each temperature which are then used as initial values for subsequent simulations.

During the production run time step is set $10^{-6}$ps. We allow system to relax another 100 ps before adding a hole into the system. By this way we make sure that, at the time a hole was added into the system, each realization has its distinct initial values. After the hole is introduced at the donor site the escape is characterized by the particle density at the donor site. If the particle density at the donor site reaches 33% of the initial density, the particle is assumed to be escaped from the donor site.

Figure 32 illustrates the phase space plot of the escape. It can be seen from the figure that the system dynamics of the system a certain escape plane. i.e. it is seen in the figure that the trajectory tries to leave from one side while on the other side it attracts to an attractor. The dynamics of the system in the phase space requires further investigation. The mechanism of escape is poorly understood. In fact the stochastic nature of the process allows different trajectories to take different path even for same initial conditions. The complex dynamics of the state variables defines the escape, while the correlation between them yet to be defined. However this interesting problem is not addressed in the present work.

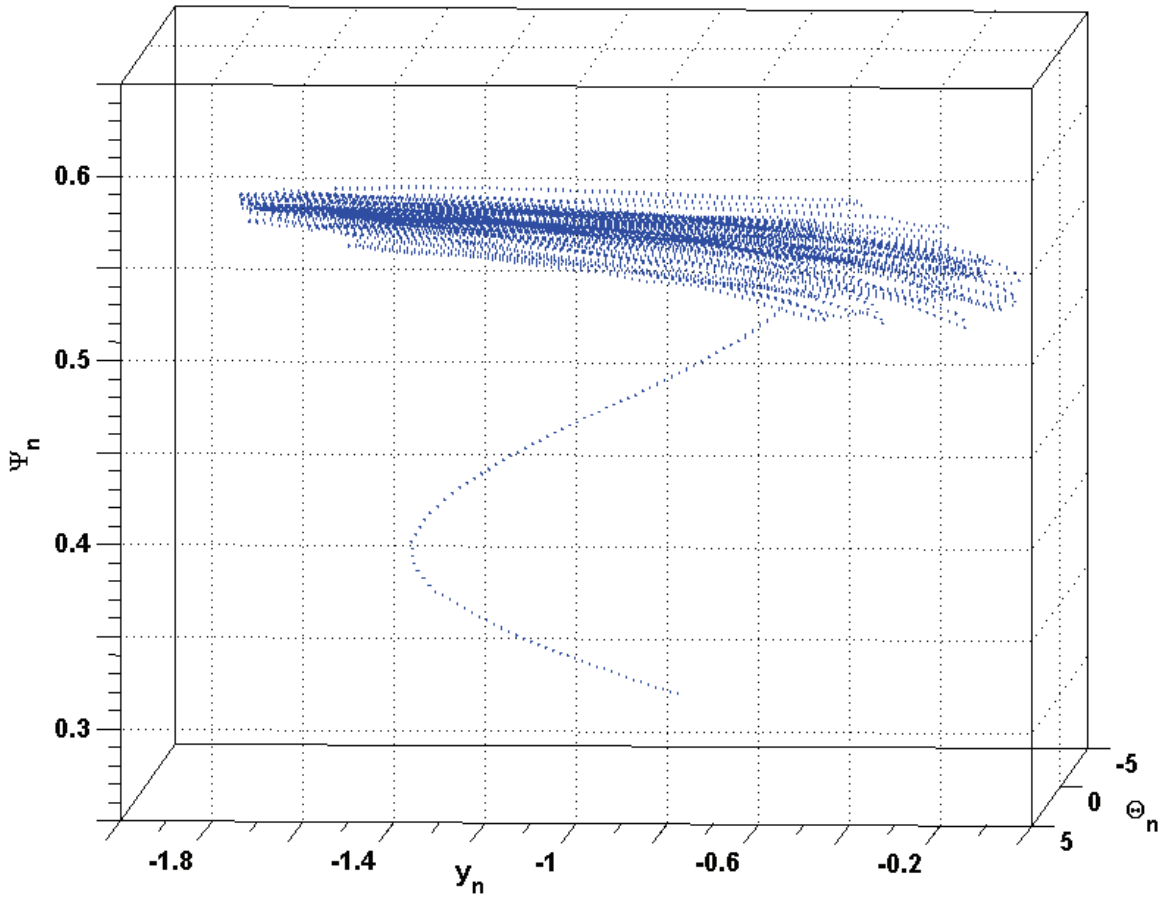


Figure 32 : System dynamics in the phase space. The trajectory diverges at the corner while converge at the other corner. Trajectory finally escapes at the divergent corner.

(a) Different mechanisms of escape in DNA

Several escape mechanisms are possible depending on the donor acceptor separation. When donor and acceptor are separated by relatively small distance the escape mainly happens through tunneling. Figure 33 shows the dependence of escape time with distance. The sharp exponential increase between first two data points represents the tunneling nature of the escape mechanism. However as distance between donor and acceptor increase further the system becomes dissipative due to the onsite energy disorder introduced by the dynamical fluctuations. For donor acceptor distance of 4 to 6 base pairs, the polaron assisted hopping also contribute to the particle

escape along with the tunneling. The escape time now shows power low dependence, see Figure 33. Finally beyond donor acceptor distance of six base pairs, the polaron assistant hoping becomes dominant. At this point the escape is almost independent of the donor-acceptor separation. The escape time now depends only on the energy depth of the trap. After escape from the donor site the particle travels across the DNA as a polaron hopping.

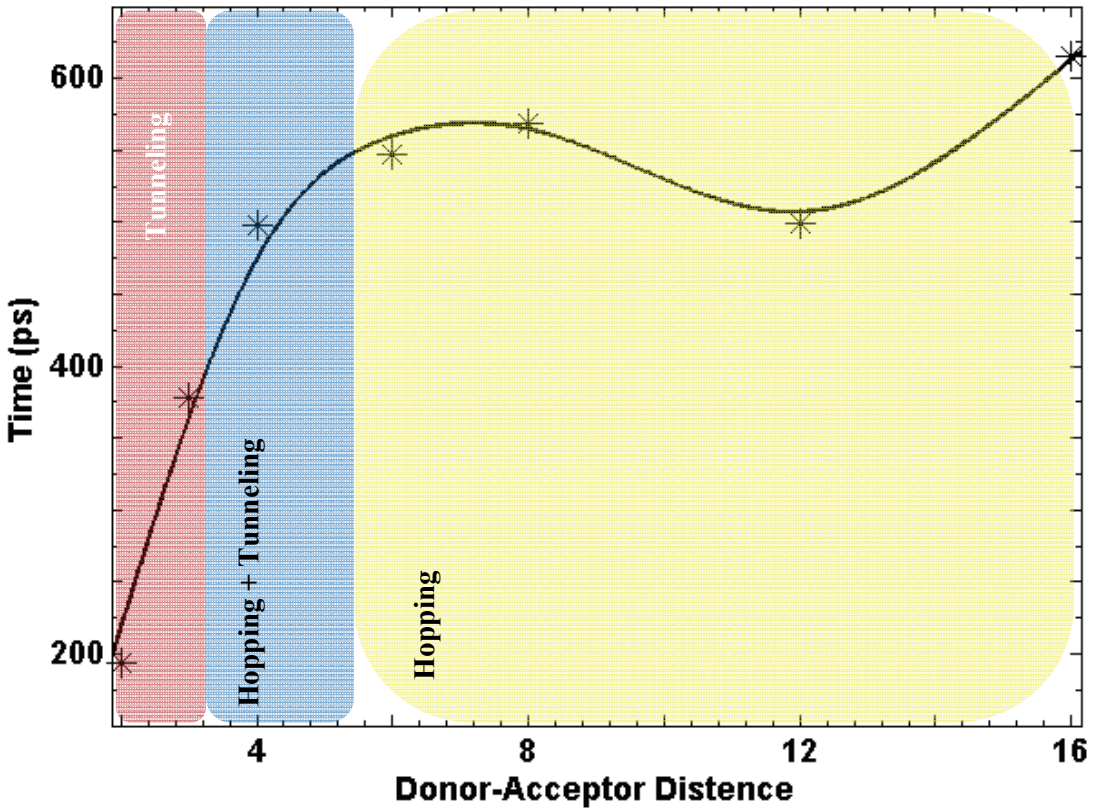


Figure 33 : dependence of escape time as a function of the donor acceptor separation at T= 60 K

(b) Polaron assisted charge hopping

In Figure 34 and Figure 35 the polaron assisted escape it illustrated. In these figures the DNA is relaxed for about 100 ps. Then at around 100 ps a particle (a hole) is introduced into the system.

As seen in the figures, introduction of a hole followed by polaron formation. This fact is clearly visible in each figure as large structural fluctuations around the donor site. However it should be noted that the two figures are different realizations of an identical system. Both figures represent a system with donor acceptor separation of 16 base pairs. It should be noted that donor site is located in the middle of the strand whereas the acceptor site is located to the left from the donor site, at 240$^{th}$ base pair. Charge lattice coupling constant is 1.8 .

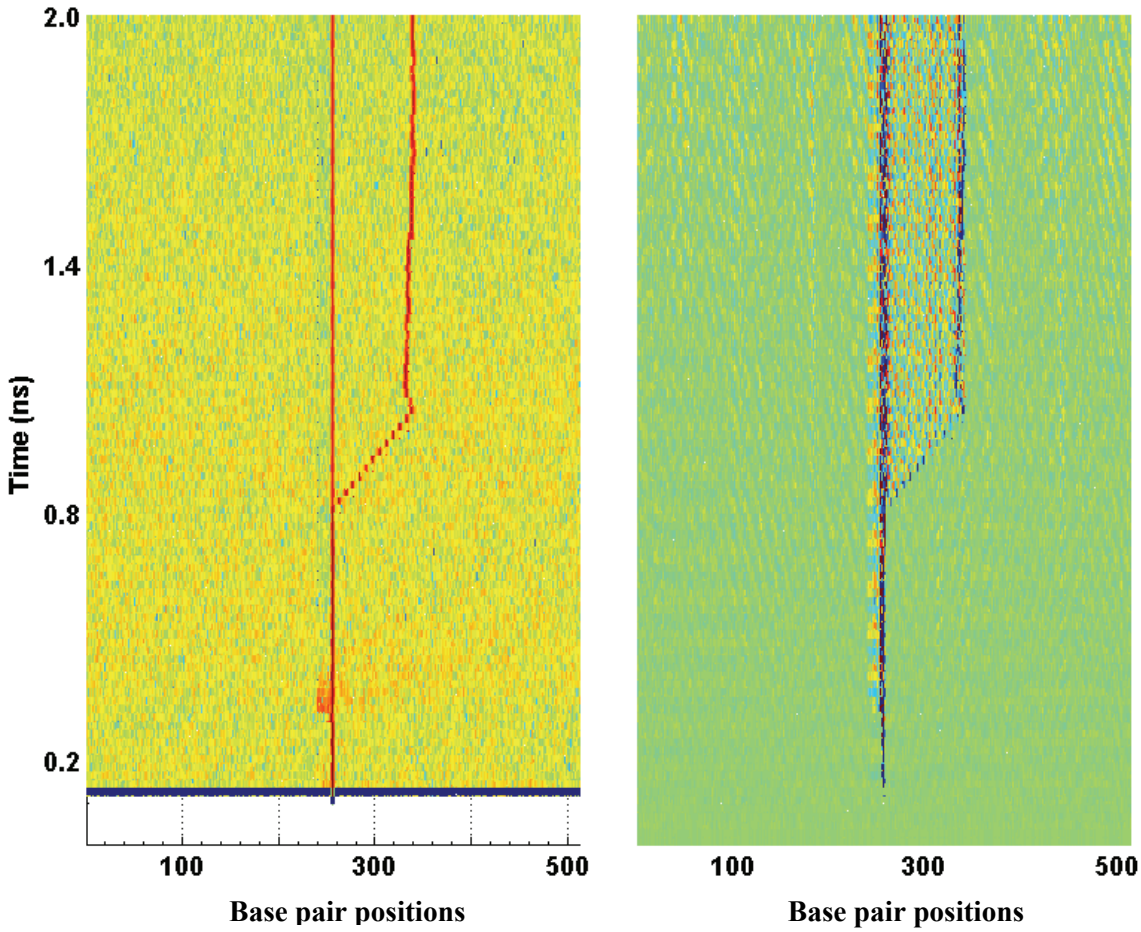


Figure 34 : Polaron formation and polaron assisted escape of trapped hole

As it was illustrated earlier, due to the stochastic nature of the system dynamics, even with the same initial conditions the system can evolve through different paths. The escapes in Figure 34

and Figure 35 are very distinct from each other. In Figure 34 the polaron moves across the DNA relatively fast compared to the Figure 35. Also it should be noted that the polaron motion is now distance independent. The hole could travel as much as 200 base pairs with the aid of polaron. Moreover the time scale of such motion is also diverse. In the Figure 34 polaron travels about 84 base pairs within 260 ps. The speed is about 1.1 $Åps^{-1}$. Whereas in the Figure 35 polaron travels about 150 base pairs within 1.62 ns. This corresponds to the speed of 0.314 $Åps^{-1}$.

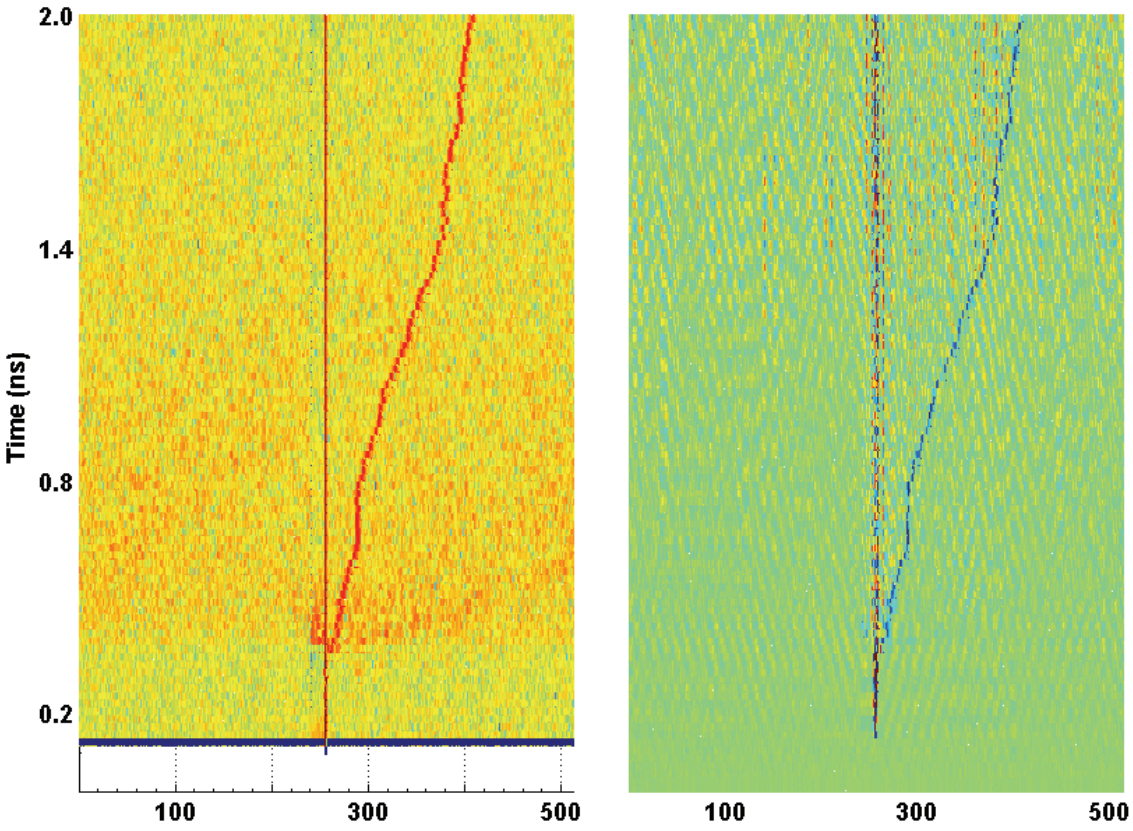


Figure 35 : Polaron formation and Polaron assisted escape

(g) Effect of charge lattice coupling on charge transfer

Even though individual realizations of a stochastic process can take different paths, an ensemble average still could yield a useful insight into the dynamics of the system. Figure 37 illustrate the

escape time as a function of temperature and the lattice charge coupling. In general the escape time depends on the two factors, which are depth of the trap which determines the height of the barrier between the donor and acceptor, and the transfer integral between the nearest neighbor base pairs, which determines the charge hopping and correspondingly the rate of tunneling through the barrier. Fluctuations in the DNA skeleton could change each of these parameters.

The onsite energy of a charge can change with variation of the local hydrogen bond stretching along the transverse direction. Once a polaron is formed this could lead to a reduction of energy of the charge. Such reduction tends to trap the charge at the site where polaron is formed. However as shown in the Figure 36, if charge is sufficiently large, the influence of the charge on the lattice could extend to the neighboring sites of the donor. i.e. a large polaron could form. This intern could change the energy profile of the system facilitating charge escape through polaron assisted hopping, in which polaron can move across the DNA. On the other hand the effective overlaps between electron clouds of adjacent bases are determined by the relative angle between them.

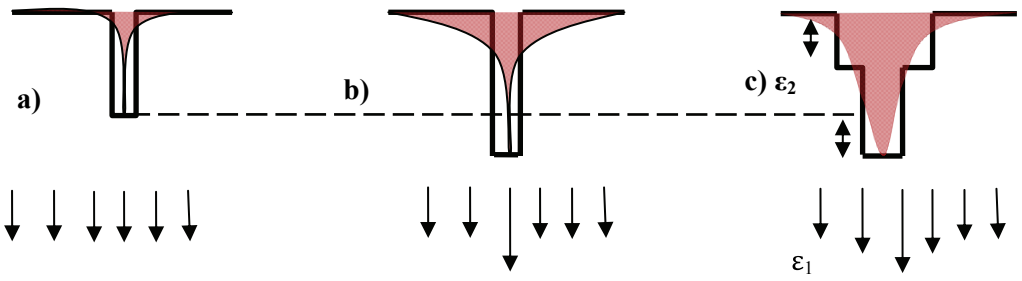


Figure 36 : Polaron assisted hopping in the figure a) GC trap is shown. b) Increase in trap depth due to formation of small polaron. c) Formation of large polaron results in the reduction of effective trap height.

The transfer integral between adjacent bases is a function of effective overlap between corresponding electron clouds, i.e. molecular orbital. Depending on the state, i.e. dynamics of the system, different transfer integrals and trap depths are possible. As shown in the Figure 37 the escape time is a complex function of temperature and charge phonon coupling. The length of the DNA is fixed at 512 base pairs. The final escape time was calculated as average over 60 different realizations of the system Each realization was realized by assigning initial values which was generated by a separate run. Each realization was carried out using the protocol mentioned in the Chapter IV.Section B.01(d).

As we have observed in our investigation, polaron formation (Kalosakas et al. 1998, Maniadis et al. 2003) in the DNA is a complex function of temperature and the phonon charge coupling constant. Tendency to polaron formation increases monotonically with the increase of phonon charge coupling. However the type of the polaron formed depends on the temperature of the system. As an example the average strength of vibrations increases with temperature. This facilitates formation of large polaron (width of the polaron is large). However if the temperature of the system is low, then the polaron is relatively small. As mentioned previously the onsite energy of the system is a liner function of the of the displacement of the site *n* as $\varepsilon_n = \varepsilon_0 + \chi y_n$ . Due to this relationship, different types of polaron contribute differently to the energy profile of the DNA strand (Maniadis et al. 2003). As an example if large polaron is formed it will affect the onsite energy of the neighboring sites too. This can reduce effective height of the trap. As a result charge can escape easily compared to the small polaron situation. In case of small polaron it only changes the energy profile locally. This introduces an additional height to the trap, causing an additional trapping of the charge.

As seen in the Figure 37, for extremely large charge phonon-coupling (phonon charge coupling value of 1.8eVÅ$^{-1}$) escape time is relatively small. This suggests that due to significantly large coupling constant, large polaron are formed at all the temperatures. This changes the characteristic of the barrier between donor and acceptor. Essentially it reduces the barrier height. There for charge can easily escape from the trap. However it should be noted that even for large phonon-charge couplings, temperature increases the width of the polaron and as a result escape time reduces.

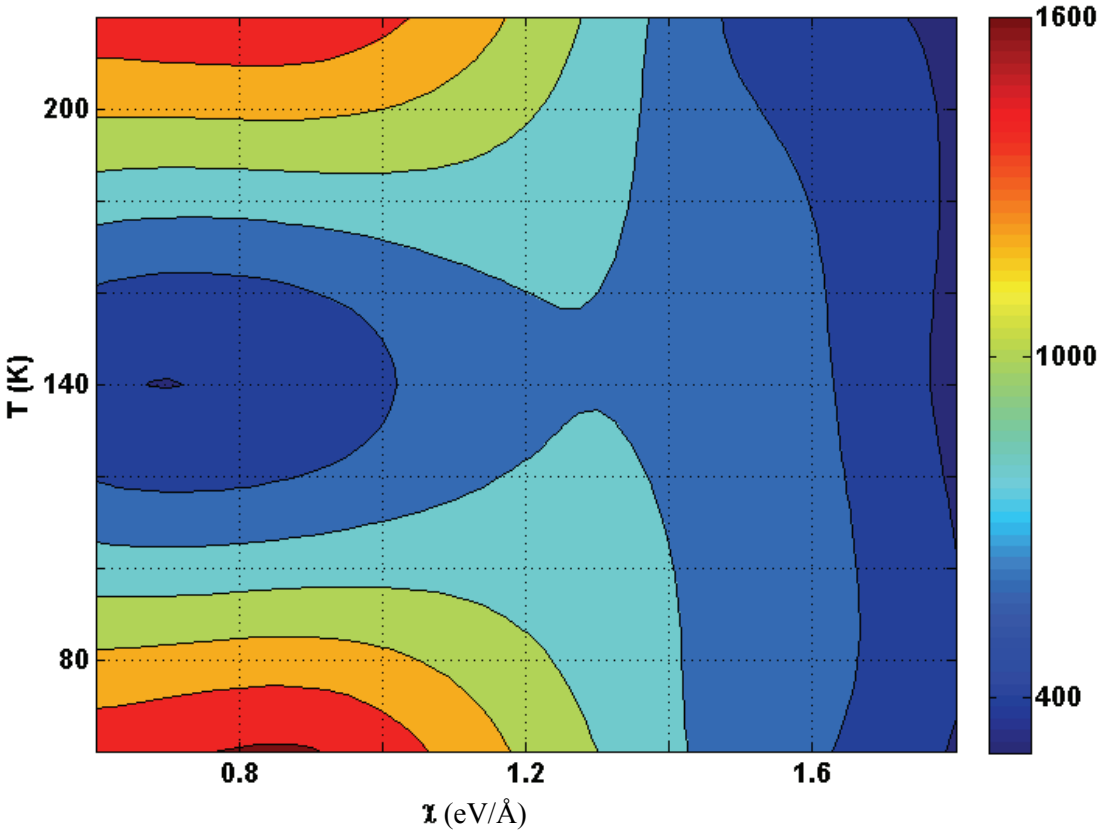


Figure 37 : escapes time as a function of temperature and charge-phonon coupling.

However at low charge phonon-coupling values there is interplay between depth of the trap and the effective width of the trap

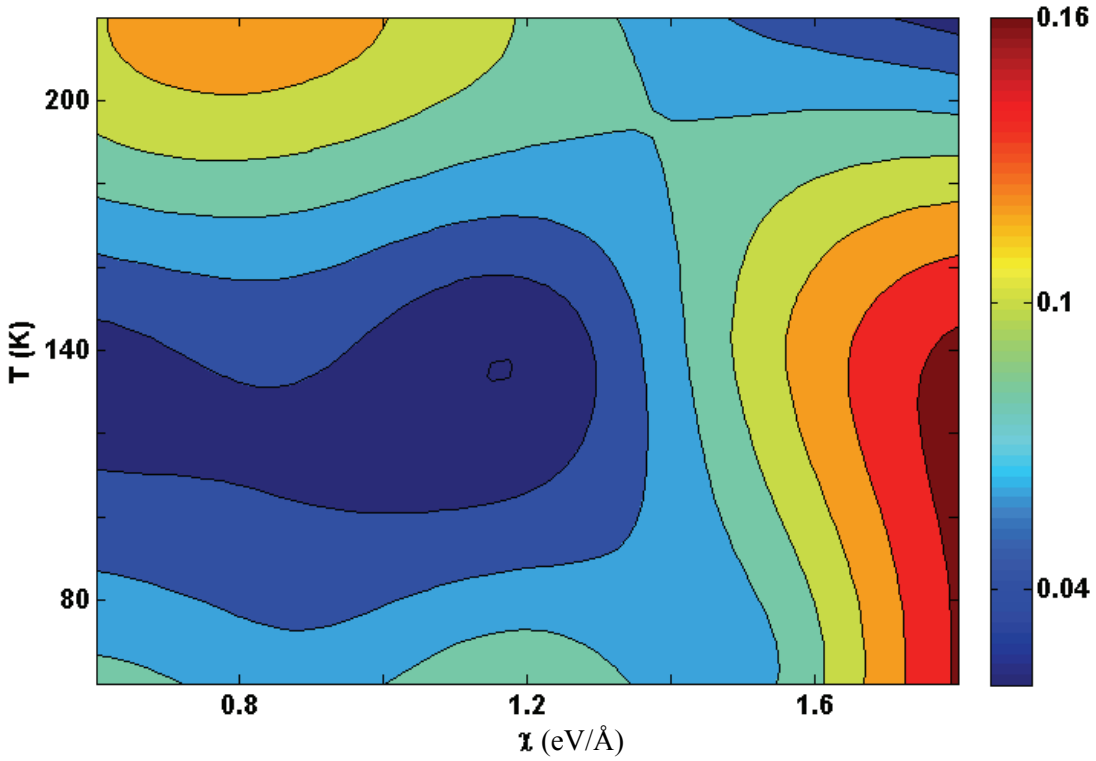


Figure 38 : Change in onsite energy at the donor site due to polaron formation

. Energy profile of the DNA consisted of an energy trap (donor site) due to the lower onsite energy of the GC WC base pair compared to the AT base pair. Polaron changes this energy trap by changing onsite energy. When a charge is introduced into the lattice it polarizes the media which stabilized the charge. This in turn reduces the energy of the charge. But now induced polarization also moves with charge creating a polaron. In order to investigate the temperature dependence on polaron formation we have computed average polaron width as,

$$d = min\left(\sum_{n=\frac{N}{2}-k}^{\frac{N}{2}+k}\sum_{t=t_{final}-t_0}^{t_{final}} y_{i,n}(t)\right) \quad (37)$$

Where k=0..10 and $t_0$ is characteristic time to identify the polaron. $t_{final}$ is the time at which charge escapes from the donor site and N is the length of the DNA. The k value for which above quantity is minimized is assumed to be the width of the polaron.

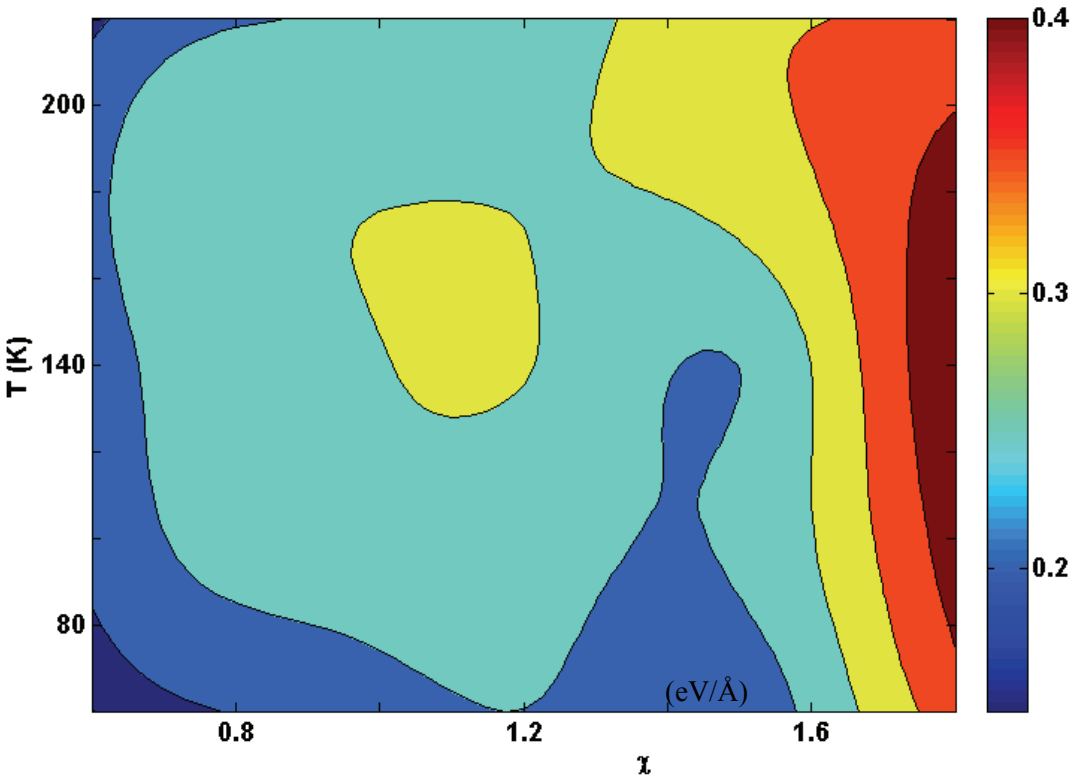


Figure 39: Average energy change on the bridge due to polaron formation

When polaron is formed it reduces the onsite energy of the system. This fact is used as a measure of polaron formations. In order to have energy drop we should have a negative y values and by calculating minimum of the above quantity the polaron width can be roughly estimated. However due to stochastic nature of the process, large number of realizations are needed in order to obtain an accurate value. But at the present time, such attempt is restricted due to computational cost associated with generating large number of realizations. Moreover charge transport experiments

on DNA are carried out at single molecular level. Due to this reason, it may not worthwhile to study ensemble averages. As already mentioned in the earlier discussions, due to the stochastic nature of the process even for the same initial conditions, system can evolve in different paths.

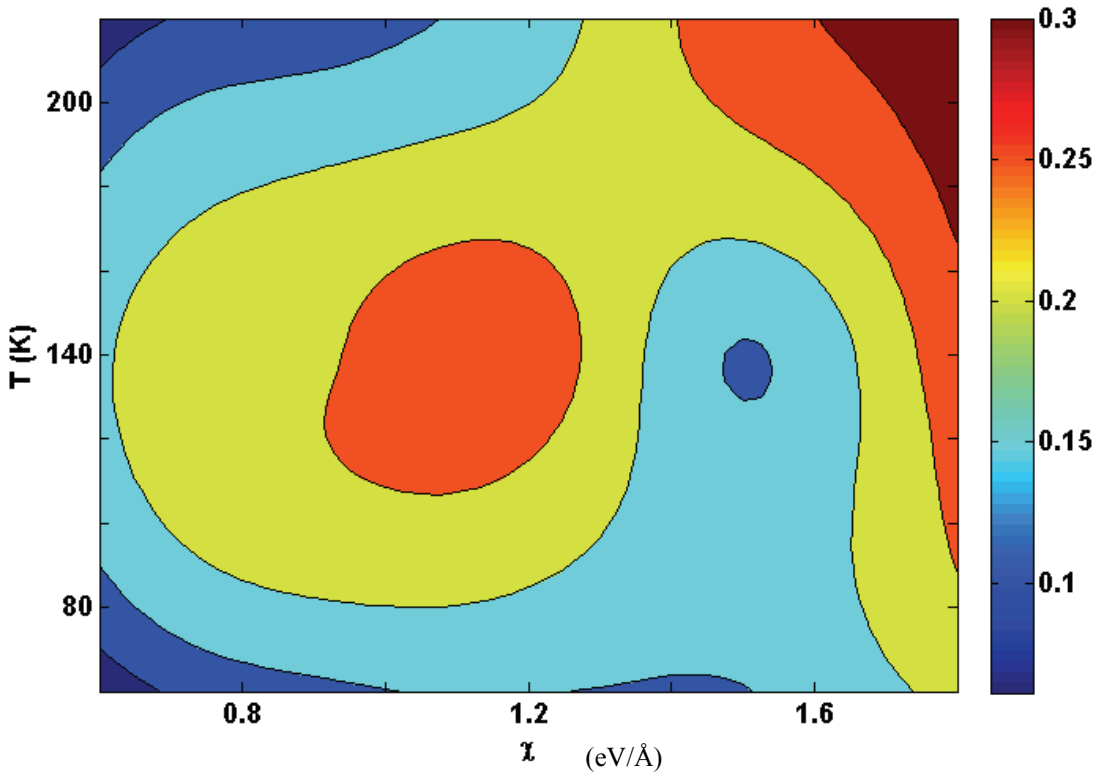


Figure 40 : Resultant change in the height of the trap due to changes in the bridge energy and the onsite energy due to polaron

Figure 38 shows the change in onsite energy due to the formation of polaron. This value corresponds to the $\varepsilon_1$ in the Figure 36. Figure 39 shows change in the well height introduced by the polaron. This corresponds to the energy $\varepsilon_2$ in the Figure 36. It could be noted that the energy $\varepsilon_2$ is the average energy drop due to the polaron formation. A charge trapped at the donor site experience different barrier height depending on the values of energy $\varepsilon_1$ and energy $\varepsilon_2$. As an

example, as illustrated in the Figure 40 the effective barrier height is reduced significantly due to polaron formation. As seen in the Figure 40 at high phonon charge coupling limit effective barrier height can be changed by around 0.3 eV. It should be noted that the barrier introduced by the GC trap is only 0.4 eV. At high phonon charge coupling limit with high temperature, a charge trapped at the donor site experience a barrier around 0.1 eV, whereas at low phonon charge coupling with low temperature region barrier almost unchanged resulting a charge trapped at a donor site to experience about 0.4 eV barrier height. It should be noted that at low phonon charge coupling limit temperature effects are not monotonic. Correspondingly the escape time is also shows non monotonic behavior. In summary effect of temperature on the charge transport properties shows very complex pattern. There are two main effects which determine the escape rate of a charge trapped at a donor site. In general, different realizations of the DNA form different type of polaron. The shape of the polaron formed is important because it changes the energy profile. For example is the small polaron if formed then it only introduce additional trapping due to increase in the height of the trap. However if a large polaron is formed it changes the effective height of the trap so that charge can move to neighboring site easily. But moving charge takes its polarization along with it, hence effectively polaron itself moves.

# Chapter 5. Conclusions:

We have studied the charge transport though DNA. I-V characteristic was studied using non equilibrium Green function method. Whereas charge transfer between donor and acceptor sites was studied solving a stochastic nonlinearly coupled differential equations system which describes the dynamics of DNA. We have found the relatively small charge transfer integrals with large energy band gap results in thin energy bands. These thin energy bands are highly vulnerable to the structural fluctuations, structural defects, environmental effect and coupling to the external objects such as leads. Furthermore our investigation reveal that in order to have good conductance through DNA, the coupling between DNA and contacts needs to be roughly equal to the charge transfer integrals between DNA bases. Both higher coupling and lower coupling results in poor conductance through DNA.

Mismatch base pair can act as defects in a DNA strand. We focused our attention on G-A mismatch base pair as it only form very little structural changes. This introduces difficulties in order to identifying G-A mismatch base pairs by rapier enzyme which relies on structural abnormalities. We have studied the effect of the G-A mismatch base pair on I-V characteristic and discovered that the presence of a mispair always reduces the current through homogeneous DNA strands. However due to the thin energy bands, G-A mismatched base pair results in either increase or decrease in current in the case of inhomogeneous DNA strand. Moreover we have found out that even though G-A mismatched base pair only introduces small structural changes it indeed changes the electronic properties of DNA for long distance. As an example we have found out that the effect of G-A mispair could be seen as far as 90 base pairs for a homogeneous DNA. Even though this value is significantly smaller in the case of inhomogeneous DNA, G-A mispair changes the electrical properties of DNA significantly. We have found out some

characteristic peaks in the probability density functions which could be used to identify the presence of mismatched base pair. In summary, current voltage characteristic could be used to identify the DNA strands with mismatched base pairs. At finite temperature DNA undergoes relatively large magnitude fluctuations. Especially these fluctuation changes the twist angel between DNA bases. Changes in twist angle directly impact on the current voltage characteristic of the DNA as charge transfer integrals are function of twist angle. We found a reduction in current as thermal fluctuations are introduced into the system. More over thermal fluctuations suppress the effects introduced by structural abnormalities, such as defects.

The fact that DNA bases are loosely bounded by either through sugar phosphate backbone or through hydrogen bonds, make DNA is vulnerable to large structural fluctuations. This highly flexible nature of the DNA support extremely long range, in fact sometimes fast, charge transport via polaron assisted hoping. We found out polaron formed in the DNA could propagate as long as 180 base pairs. However the formation of a polaron itself is a complex process. We found that due to stochastic nature of the system, not all realizations formed polarons even with same initial conditions.

We have found very complex dependence of charge transport properties on charge lattice coupling strength and the temperature. These complex dependencies along with the fact that DNA charge transport is a stochastic process give rise to highly diverse outcomes of DNA charge transport experiments. In fact experimental researches have reported counterintuitive results on DNA charge transport in the literature. As most of the time these experiments are performed in a single molecule level, the diversity of these results could be attribute to the stochastic nature and the complex structural dynamics of the DNA. We have performed systematic study to investigate charge escape from GC site as a function of temperature and

phonon charge coupling constant. The exact value of phonon charge coupling constant is unknown. In fact this value depends on the nature of the environment at which electron is introduced. For soft molecule like DNA structure factors like atomic distance can be changed significantly by introducing strain into the DNA. In such case phonon charge coupling could be varied which inturn facilitate different charge dynamics.

# Chapter 6. Future Directions

## (a) Charge transport through DNA: Phase incoherent case

In the present investigation we have only considered I-V characteristic of the DNA under phase coherence limit. We have assumed that a charge moving through the DNA doesn't interact with any other electrons nor with lattice vibrations (phonons). However this assumption is weak within the framework of the DNA as DNA undergoes high structural fluctuations and is subject to various phonon modes. On the other hand phase incoherence treatment is possible within non equilibrium green function method (Anantram et al. 2008). As an extension to the current study I would like to perform IV characteristic in phase incurrence limit.

## (b) Environmental influence on charge transport properties in DNA

As mentioned in the introductory chapter environmental effects can influence charge transport properties via changing onsite energies of the DNA. Even though this problem requires all atom molecular dynamic simulations, an insight into the problem can be gained via mean field continuum approach described in the Chapter IV.Section 2.02. Now, in order to find the Eigen values of the system we have to solve nonlinear Eigen value equations where it's Hamiltonian itself depends on the energy of the system.

## (c) Parameters for mean field computations

Parameters for the coarse grained models need to be computed using first principle computations. In the present investigation we have used parameters reported by different research groups. However parameters are not available for mismatched base pairs such as G-A mispair. specially, dependence of charge transfer integral on the twist angle is not available for mispair. I would like to compute these parameters which can be used with tight binding.

(d) All atom QM/MM simulations to study polaron formation in DNA

In order accurately describe the polaron formation in DNA at first principle level; one may need to utilize Quantum Mechanics - Molecular Mechanics (QM/MM) hybrid approach (Kamerlin, Haranczyk and Warshel 2008, Warshel and Levitt 1976). Particularly such treatment is inherently difficult due to extended size of the molecule. However it has been shown that, within the continuum mean field approach, a motion of a polaron be could controlled by an external potential (Zheng et al. 2006). It would be interesting to investigate such problem at the first principle level.

References.


1. Almbladh, C. O. & et al. (1983) Accuracy of the Hartree-Fock and Local Density Approximations for Electron Densities: A study for Light Atoms. *Physica Scripta,* 28**,** 389.
2. Almbladh, C. O. & A. C. Pedroza (1984) Density-functional exchange-correlation potentials and orbital eigenvalues for light atoms. *Physical Review A,* 29**,** 2322.
3. Anantram, M. P., M. S. Lundstrom & D. E. Nikonov (2008) Modeling of Nanoscale Devices. *Proceedings of the IEEE,* 96**,** 1511-1550.
4. Anderson, P. W. (1958) Absence of Diffusion in Certain Random Lattices. *Physical Review,* 109**,** 1492.
5. Barbi, M., S. Cocco & M. Peyrard (1999) Helicoidal model for DNA opening. *Physics Letters A,* 253**,** 358-369.
6. Berlin, Y. A., A. L. Burin & M. A. Ratner (2000) DNA as a molecular wire. *Superlattices and Microstructures,* 28**,** 241-252.
7. Bloch, F. (1929) Über die Quantenmechanik der Elektronen in Kristallgittern. *Zeitschrift für Physik A Hadrons and Nuclei,* 52**,** 555-600.
8. Boon, E. M. & J. K. Barton (2002) Charge transport in DNA. *Current Opinion in Structural Biology,* 12**,** 320-329.
9. Braun, E., Y. Eichen, U. Sivan & G. Ben-Yoseph (1998) DNA-templated assembly and electrode attachment of a conducting silver wire. *Nature,* 391**,** 775-778.
10. Brown, T., G. A. Leonard, E. D. Booth & J. Chambers (1989) Crystal structure and stability of a DNA duplex containing A(anti) · G(syn) base-pairs. *Journal of Molecular Biology,* 207**,** 455-457.
11. Cai, L., H. Tabata & T. Kawai (2000) Self-assembled DNA networks and their electrical conductivity. *Applied Physics Letters,* 77**,** 3105-3106.
12. Cuniberti, G., L. Craco, D. Porath & C. Dekker (2002) Backbone-induced semiconducting behavior in short DNA wires. *Physical Review B,* 65.

13. Dandliker, P. J., R. E. Holmlin & J. K. Barton (1997) Oxidative Thymine Dimer Repair in the DNA Helix. *Science,* 275**,** 1465-1468.

14. Davidson, E. R. & D. Feller (1986) Basis set selection for molecular calculations. *Chemical Reviews,* 86**,** 681-696.

15. de Moura, F. A. B. F. & M. L. Lyra (1998) Delocalization in the 1D Anderson Model with Long-Range Correlated Disorder. *Physical Review Letters,* 81**,** 3735.

16. de Pablo, P. J., F. Moreno-Herrero, J. Colchero, oacute, J. mez Herrero, P. Herrero, Bar, A. M., Ordej, P. n, J. Soler, eacute, M & E. Artacho (2000) Absence of dc-Conductivity in lambda -DNA. *Physical Review Letters,* 85**,** 4992.

17. Debrabant, K. & A. Rößler (2009) Families of efficient second order Runge-Kutta methods for the weak approximation of Itô stochastic differential equations. *Applied Numerical Mathematics,* 59**,** 582-594.

18. Drew, H. R., R. M. Wing, T. Takano, C. Broka, S. Tanaka, K. Itakura & R. E. Dickerson (1981) Structure of a B-DNA dodecamer: conformation and dynamics. *Proceedings of the National Academy of Sciences of the United States of America,* 78**,** 2179-2183.

19. Edirisinghe, N., V. Apalkov, J. Berashevich & T. Chakraborty (2010) Electrical current through DNA containing mismatched base pairs. *Nanotechnology,* 21**,** 245101.

20. Eley, D. D. & D. I. Spivey (1962) Semiconductivity of organic substances. Part 9.-Nucleic acid in the dry state. *Transactions of the Faraday Society,* 58**,** 411-415.

21. Emberly, E. G. & G. Kirczenow (1998) Theoretical study of electrical conduction through a molecule connected to metallic nanocontacts. *Physical Review B,* 58**,** 10911.

22. Endres, R. G., D. L. Cox & R. R. P. Singh (2004) Colloquium: The quest for high-conductance DNA. *Reviews of Modern Physics,* 76**,** 195.

23. Fink, H. W. & C. Schonenberger (1999) Electrical conduction through DNA molecules. *Nature,* 398**,** 407-410.

24. Fixe, F., V. Chu, D. M. F. Prazeres & J. P. Conde (2005) Single base mismatch detection by microsecond voltage pulses. *Biosensors and Bioelectronics,* 21**,** 888-893.
25. Franklin R. & R. G. Gosling (1953) Molecular Configuration in Sodium Thymonucleate *Nature,* 171**,** 740.
26. Gikhman I. , S. A. 1972. *Stochastic differential equations*. Berlin.
27. Gollisch, H. & L. Fritsche (1981) Density matrix calculations for molecules and clusters I. Theoretical foundations. *Journal of Physics B: Atomic and Molecular Physics,* 14**,** 4441.
28. Hall, D. B., R. E. Holmlin & J. K. Barton (1996) Oxidative DNA damage through long-range electron transfer. *Nature,* 382**,** 731-735.
29. Hanks, P. & J. C. Stoddart (1979) Lattice effects in the ferromagnetic response functions. *Journal of Physics C: Solid State Physics,* 12**,** 3025.
30. Hariharan, P. C. & J. A. Pople (1973) The influence of polarization functions on molecular orbital hydrogenation energies. *Theoretical Chemistry Accounts: Theory, Computation, and Modeling (Theoretica Chimica Acta),* 28**,** 213-222.
31. Hedin, L. & B. I. Lundqvist (1971). *J. Phys. C: Solid St. Phys.,* 4**,** 2064.
32. Hehre, W. J., R. F. Stewart & J. A. Pople (1969) Self-Consistent Molecular-Orbital Methods. I. Use of Gaussian Expansions of Slater-Type Atomic Orbitals. *The Journal of Chemical Physics,* 51**,** 2657-2664.
33. Heim, T., D. Deresmes & D. Vuillaume (2004) Conductivity of DNA probed by conducting--atomic force microscopy: Effects of contact electrode, DNA structure, and surface interactions. *Journal of Applied Physics,* 96**,** 2927-2936.
34. Helgren E., A. Omerzu , G. Gr¨uner, D. Mihailovic, R. Podgornik & H. Grimm (2001) Electrons on the double helix: optical experiments on native DNA. *arXiv:cond-mat/0111299v1*.
35. Hiroaki, Y. & I. Kazumoto (2010) Some Effective Tight-Binding Models for Electrons in DNA Conduction: A Review. *Advances in Condensed Matter Physics,* 2010**,** 1.

36. Hodzic, V. & R. W. Newcomb (2007) Modeling of the Electrical Conductivity of DNA. *Circuits and Systems I: Regular Papers, IEEE Transactions on,* 54**,** 2360-2364.

37. Hohenberg, P. & W. Kohn (1964) Inhomogeneous Electron Gas. *Physical Review,* 136**,** B864.

38. Hwang, J. S., K. J. Kong, D. Ahn, G. S. Lee, D. J. Ahn & S. W. Hwang (2002) Electrical transport through 60 base pairs of poly(dG)-poly(dC) DNA molecules. *Applied Physics Letters,* 81**,** 1134-1136.

39. Joe, Y. S., S. H. Lee & E. R. Hedin (2010) Electron transport through asymmetric DNA molecules. *Physics Letters A,* 374**,** 2367-2373.

40. Jones, R. O. & O. Gunnarsson (1989) The density functional formalism, its applications and prospects. *Reviews of Modern Physics,* 61**,** 689.

41. Joy, A. & G. B. Schuster (2005) Long-range radical cation migration in DNA: Investigation of the mechanism. *Chemical Communications***,** 2778-2784.

42. Kalosakas, G., S. Aubry & G. P. Tsironis (1998) Polaron solutions and normal-mode analysis in the semiclassical Holstein model. *Physical Review B,* 58**,** 3094.

43. Kalosakas, G., K. L. Ngai & S. Flach (2005) Breather-induced anomalous charge diffusion. *Physical Review E,* 71**,** 061901.

44. Kamerlin, S. C. L., M. Haranczyk & A. Warshel (2008) Progress in Ab Initio QM/MM Free-Energy Simulations of Electrostatic Energies in Proteins: Accelerated QM/MM Studies of pKa, Redox Reactions and Solvation Free Energies†. *The Journal of Physical Chemistry B,* 113**,** 1253-1272.

45. Kasdin, J. (1995a) Discrete Simulation of Colored Noise and Stochastic Processes and 1/f^a Power Law Noise Generation. *Proceedings of the IEEE,* 83**,** 802.

46. Kasdin, J. (1995b) Runge-Kutta algorithm for the numerical integration of stochastic differential equations, . *Journal of Guidance, Control, and Dynamics,,* 18**,** 114.

47. Kasumov, A. Y., M. Kociak, S. Gueron, B. Reulet, V. T. Volkov, D. V. Klinov & H. Bouchiat (2001) Proximity-Induced Superconductivity in DNA. *Science,* 291**,** 280-282.

48. Kelley, S. O., N. M. Jackson, M. G. Hill & J. K. Barton (1999) Long-Range Electron Transfer through DNA Films. *Angewandte Chemie International Edition,* 38**,** 941-945.

49. Kohn, W. & L. J. Sham (1965) Self-Consistent Equations Including Exchange and Correlation Effects. *Physical Review,* 140**,** A1133.

50. Komineas, S., G. Kalosakas & A. R. Bishop (2002) Effects of intrinsic base-pair fluctuations on charge transport in DNA. *Physical Review E,* 65**,** 061905.

51. Konno, K., M. Nishida, S. Tanda & N. Hatakenaka (2007) The effect of dissipation on quantum transmission resonance. *Physics Letters A,* 368**,** 442-449.

52. Krompiewski, S. (1998) Ab initio studies of Ni-Cu-Ni trilayers: layer-projected densities of states and spin-resolved photoemission spectra. *Journal of Physics: Condensed Matter,* 10**,** 9663.

53. Kubar□, T. s., P. B. Woiczikowski, G. Cuniberti & M. Elstner (2008) Efficient Calculation of Charge-Transfer Matrix Elements for Hole Transfer in DNA. *The Journal of Physical Chemistry B,* 112**,** 7937-7947.

54. Langreth & et al. (1981) Easily Implementable Nonlocal Exchange-Correlation Energy Functional. *Physical Review Letters,* 47**,** 446.

55. Langreth & et al. (1983) Beyond the local-density approximation in calculations of ground-state electronic properties. *Physical Review B,* 28**,** 1809.

56. Linak, M. C. & K. D. Dorfman (2010) Analysis of a DNA simulation model through hairpin melting experiments. *The Journal of Chemical Physics,* 133**,** 125101-9.

57. MacDonald, A. H. & et al. (1980) Density functional approximation for the quasiparticle properties of simple metals. I. Theory and electron gas calculations. *Journal of Physics F: Metal Physics,* 10**,** 1719.

58. Maciá, E. & S. Roche (2006) Backbone-induced effects in the charge transport efficiency of synthetic DNA molecules. *Nanotechnology,* 17**,** 3002-3007.

59. Maniadis, P., G. Kalosakas, K. Rasmussen, Oslash & A. R. Bishop (2003) Polaron normal modes in the Peyrard-Bishop-Holstein model. *Physical Review B,* 68**,** 174304.

60. Marra, G. & P. Schär (1999) Recognition of DNA alterations by the mismatch repair system. *Biochem. J.,* 338**,** 1-13.

61. Maruyama, G. (1955) Continuous Markov processes and stochastic equations. *Rend Circ Math Palermo,* 4**,** 48.

62. Matthew A. Young, G. R., and D. L. Beveridge (1997) A 5-Nanosecond Molecular Dynamics Trajectory for B-DNA: Analysis of

63. Structure, Motions, and Solvation. *Biophysical Journal,* 73**,** 2313.

64. Mehrez, H. & M. P. Anantram (2005) Interbase electronic coupling for transport through DNA. *Physical Review B,* 71**,** 115405.

65. Mohn, P. & K. Schwarz (1993) Supercell calculations for transition metal impurities in palladium. *Journal of Physics: Condensed Matter,* 5**,** 5099.

66. Moran, D., A. C. Simmonett, F. E. Leach, W. D. Allen, P. v. R. Schleyer & H. F. Schaefer (2006) Popular Theoretical Methods Predict Benzene and Arenes To Be Nonplanar. *Journal of the American Chemical Society,* 128**,** 9342-9343.

67. Murphy, C., M. Arkin, Y. Jenkins, N. Ghatlia, S. Bossmann, N. Turro & J. Barton (1993) Long-range photoinduced electron transfer through a DNA helix. *Science,* 262**,** 1025-1029.

68. Newton, M. D., W. A. Lathan, W. J. Hehre & J. A. Pople (1969) Self-Consistent Molecular-Orbital Methods. III. Comparison of Gaussian Expansion and PDDO Methods Using Minimal STO Basis Sets. *The Journal of Chemical Physics,* 51**,** 3927-3932.

69. Norman, M. R. (1986) Model supercell local-density calculations of the 3 excitation spectra in NiO. *Physical Review B,* 33**,** 8896.

70. Ohshiro, T. & Y. Umezawa (2006) Complementary base-pair-facilitated electron tunneling for electrically pinpointing complementary nucleobases. *Proceedings of the National Academy of Sciences of the United States of America,* 103**,** 10-14.

71. Oppenheimer, M. B. a. R. (1927). *Ann. Phys. (Leipzig)* 84**,** 457.

72. Porath, D., A. Bezryadin, S. de Vries & C. Dekker (2000) Direct measurement of electrical transport through DNA molecules. *Nature,* 403**,** 635-638.

73. Porath, D., G. Cuniberti & R. Di Felice. 2004. Charge Transport in DNA-Based Devices. In *Long-Range Charge Transfer in DNA II,* ed. G. B. Schuster, 183-228. Springer Berlin / Heidelberg.

74. Porezag, D., T. Frauenheim, ouml, T. hler, G. Seifert & R. Kaschner (1995) Construction of tight-binding-like potentials on the basis of density-functional theory: Application to carbon. *Physical Review B,* 51**,** 12947.

75. Ratner, M. (1999) Photochemistry: Electronic motion in DNA. *Nature,* 397**,** 480-481.

76. Roche, S. (2003) Sequence Dependent DNA-Mediated Conduction. *Physical Review Letters,* 91.

77. Rößler, A. (2009) Second Order Runge–Kutta Methods for It◎ Stochastic Differential Equations. *SIAM Journal on Numerical Analysis,* 47**,** 1713.

78. Rumelin, W. (1982) Numerical treatment of stochastic differential equations. *SIAM J Num Anal,* 19**,** 604.

79. Santos, B., L. P. Viana, M. L. Lyra & F. A. B. F. de Moura (2006) Diffusive, super-diffusive and ballistic transport in the long-range correlated 1D Anderson model. *Solid State Communications,* 138**,** 585-589.

80. Schuchardt, K. L., B. T. Didier, T. Elsethagen, L. Sun, V. Gurumoorthi, J. Chase, J. Li & T. L. Windus (2007) Basis Set Exchange: A Community Database for Computational Sciences. *Journal of Chemical Information and Modeling,* 47**,** 1045-1052.

81. Slater, J. C. (1930) Atomic Shielding Constants. *Physical Review,* 36**,** 57.

82. Slater, J. C. & G. F. Koster (1954) Simplified LCAO Method for the Periodic Potential Problem. *Physical Review,* 94**,** 1498.

83. Smith, S. B., Y. Cui & C. Bustamante (1996) Overstretching B-DNA: The Elastic Response of Individual Double-Stranded and Single-Stranded DNA Molecules. *Science,* 271**,** 795-799.

84. Song, B., M. Elstner & G. Cuniberti (2008) Anomalous Conductance Response of DNA Wires under Stretching. *Nano Letters,* 8**,** 3217-3220.

85. Sonmezoglu, S., O. A. Sonmezoglu, x015F, G. Cankaya, x, ld, A. m & N. Serin (2010) Electrical characteristics of DNA-based metal-insulator-semiconductor structures. *Journal of Applied Physics,* 107**,** 124518-6.

86. Stoddart, J. C. & P. Hanks (1978) Bloch wave effects in the density response function. *Journal of Physics C: Solid State Physics,* 11**,** 1813.

87. Storm, A. J., J. van Noort, S. de Vries & C. Dekker (2001) Insulating behavior for DNA molecules between nanoelectrodes at the 100 nm length scale. *Applied Physics Letters,* 79**,** 3881-3883.

88. Tsukamoto, T., Y. Ishikawa, Y. Sengoku & N. Kurita (2009) A combined DFT/Green's function study on electrical conductivity through DNA duplex between Au electrodes. *Chemical Physics Letters,* 474**,** 362-365.

89. Uday, G. & A. K. Rajagopal (1980) Exchange-correlation potential for inhomogeneous electron systems at finite temperatures. *Physical Review A,* 22**,** 2792.

90. Voityuk, A. A. (2005) Charge transfer in DNA: Hole charge is confined to a single base pair due to solvation effects. *The Journal of Chemical Physics,* 122**,** 204904-4.

91. Vosko & et al. (1980) Influence of an improved local-spin-density correlation-energy functional on the cohesive energy of alkali metals. *Physical Review B,* 22**,** 3812.

92. Wan, C., T. Fiebig, O. Schiemann, J. K. Barton & A. H. Zewail (2000) Femtosecond direct observation of charge transfer between bases in DNA. *Proceedings of the National Academy of Sciences of the United States of America,* 97**,** 14052-14055.

93. Warshel, A. & M. Levitt (1976) Theoretical studies of enzymic reactions: Dielectric, electrostatic and steric stabilization of the carbonium ion in the reaction of lysozyme. *Journal of Molecular Biology,* 103**,** 227-249.

94. Watson J.D. & F. H. C. Crick (1953) A Structure for Deoxyribose Nucleic Acid. *Nature* 171**,** 737.

95. Wigner, E. P. (1938). *Trans. Faraday Soc.,* 34**,** 678.

96. Wilkins M.H.F., A. R. S. A. R. W., H.R. (1953) Molecular Structure of Deoxypentose Nucleic Acids. *Nature,* 171**,** 738.

97. Xu, Zhang, Li & Tao (2004) Direct Conductance Measurement of Single DNA Molecules in Aqueous Solution. *Nano Letters,* 4**,** 1105-1108.

98. Xu, M., R. Endres & Y. Arakawa (2007) The Electronic Properties of DNA Bases. *Small,* 3**,** 1539-1543.

99. Yoo, K. H., D. H. Ha, J. O. Lee, J. W. Park, J. Kim, J. J. Kim, H. Y. Lee, T. Kawai & H. Y. Choi (2001) Electrical Conduction through Poly(dA)-Poly(dT) and Poly(dG)-Poly(dC) DNA Molecules. *Physical Review Letters,* 87**,** 198102.

100. Zhang, Y., Y. Lu, J. Hu, X. Kong, B. Li, G. Zhao & M. Li (2002) Direct detection of mutation sites on stretched DNA by atomic force microscopy. *Surface and Interface Analysis,* 33**,** 122-125.

101. Zheng, B., J. Wu, W. Sun & C. Liu (2006) Trapping and hopping of polaron in DNA periodic stack. *Chemical Physics Letters,* 425**,** 123-127.

102. Zhu, Y., C.-C. Kaun & H. Guo (2004) Contact, charging, and disorder effects on charge transport through a model DNA molecule. *Physical Review B,* 69**,** 245112.

## Appendix A. Stochastic Differential Equations

Almost each and every process exposit in the nature are vulnerable to the noise. Often assumptions are made in order to simplify the problems and system equations are generated without takeing the noise into the account. However in the study of the DNA, both because the system dimension and the flexibility of the DNA the thermal noise becomes a significant factor which may not be safely disregard. We have studied the thermal effects on DNA by introducing the temperature depended random force into the system equations. This additional small term now change the solution and related technique significantly in many prospective. A deterministic ordinary differential equation will produce an accurate solution for a given initial conditions. This solution will be unique for a given initial conditions. However when a stochastic process is present solution can evolve in different path even for a given initial conditions (Rößler 2009). This has an important consequence. For a deterministic ODE the accuracy of the solution can be improved by using higher order algorithms. In general a Runge-Kutta order four algorithm is considered as a standard algorithm for ODE. But for stochastic ordinary differential equation solvers the use of higher order algorithms may not be effective as the initial conditions itself are chosen from a random distribution. Generally order two stochastic ODE solvers are believed to be sufficient. However for the system equation describing DNA it is necessary to use higher order ODE solves as in general the system tend to destabilized even with small error. For this reason we have used Runge-Kutta order four stochastic ODE solvers (Debrabant and Rößler 2009, Kasdin 1995b, Kasdin 1995a).

The common example for a stochastic process is the typical diffusion process. During the diffusion process a particle undergoes a random motion called Brownian motion. This is given by;

$$dy = a(y,t)y + b(y,t)dW_s \tag{38}$$

The first term is a deterministic term whereas the second term is a stochastic term. Notice that the in the above equation, SDE has given as a differential form. This is because many stochastic processes are non differentiable, though continuous.

A process such as Brownian motion could be mathematically characterized by the Wiener process. A Wiener process has following properties.

1. $W_t \in N(0,t)$
2. For each $t_1 < t_2$, the normal random variable $W_{t_1} - W_{t_2}$ is independent of the random variables $W_t$ $0 \leq t \leq t_1$
3. Wiener process can be represented by a continuous path

The starting point in solving an SDE is to define a chain rule for a stochastic differential, which is given by;

$$dy = \frac{\partial f(x,t)}{\partial t}dt + \frac{\partial f(x,t)}{\partial x}dW_t + \frac{1}{2}\frac{\partial^2 f(x,t)}{\partial x^2}dt \tag{39}$$

Here $y = f(x,t)$ is a stochastic process. This is called Ito formula.

$$dy(t) = a(t,y)dt + b(t,y)dW_t \tag{40}$$

$$y(c) = y_c \tag{41}$$

First step toward the development of a solution method is to use Euler's method. The SDE analogy to the to the Euler's method is called Euler's-Maruyama method (Maruyama 1955). For a SDE given by equation 1 the Euler's-Maruyama method can be defined as;

$$\omega_0 = y_0 \tag{42}$$

$$\omega_{i+1} = \omega_i + a(t_i, \omega_i)\Delta t_{i+1} + b(t_i, \omega_i)\Delta W_{i+1} \tag{43}$$

$$\Delta t_{i+1} = t_{i+1} - t_i \tag{44}$$

$$\Delta W_{i+1} = W(t_{i+1}) - W(t_i) \tag{45}$$

The Euler's-Maruyama method is order 0.5 method (Gikhman I. 1972) with respect to stochastic process. Due to this reason it lacks in accuracy. Even though higher order methods such as Milstein method can be generated from stochastic Taylor expansion this methods are seldom used in practice due to the necessity to provide partial derivatives explicitly. The common alternative used in ODE is Runge-Kutta methods. The order 1 Runge-Kutta methods are presented in equation 1 (Rumelin 1982).

$$\omega_0 = y_0 \tag{46}$$

$$\omega_{i+1} = \omega_i + a(\omega_i)\Delta t_i + b(\omega)\Delta W_i + \frac{1}{2}\left[b\left(\omega_i + b(\omega)\sqrt{\Delta t_i}\right) - b(\omega_i)\right](\Delta W_i^2 - \Delta t_i) \tag{47}$$

The random variables $\Delta W_i$ can compute as;

$$\Delta W_i = z_i\sqrt{\Delta t_i} \tag{48}$$

Where $z_i$is chosen from N (0, 1).

Appendix B. Introduction to first principle calculations:

The material properties of an atomic ensemble, a crystal or molecule such as a DNA base, is mainly determined by the behavior of valence electrons of the system. Valence electrons are the outermost electrons which are energetically active compares to inter electrons, called core electrons. Core electrons, in general, are paired and occupy filled orbital. During normal, low energy circumstances, only valence electrons contribute to the charge transport through many particle systems.

Neutral material composes of equal amount of positive charge particles (nuclei) and negative charge particles (electrons). But proton mass is almost 1000 times larger than that of electrons. Since both these particles carry same charge, forces acting on them are same. Any nuclear movement is then followed by the electron instantaneously. This enables us to treat the nuclear coordinate in adiabatic limit and facilitates the separation of electronic coordinate from the nuclear coordinate. This adiabatic principle, called Born-Oppenheimer principle (Oppenheimer 1927), enables treating the dynamics of the electron in frozen nuclear coordinate system.

In general a Hamiltonian for many body systems can be written as;

$$\hat{H} = \left[\hat{T} + \hat{V} + \hat{U}\right] = \left[\sum_{i}^{N} -\frac{\hbar^2}{2m}\nabla_i^2 + \sum_{i}^{N} V(\vec{r}_i) + \sum_{i<j}^{N} U(\vec{r}_i, \vec{r}_j)\right] \tag{49}$$

$$\left[\sum_{i}^{N} -\frac{\hbar^2}{2m}\nabla_i^2 + \sum_{i}^{N} V(\vec{r}_i) + \sum_{i<j}^{N} U(\vec{r}_i, \vec{r}_j)\right]\Psi = E\Psi \tag{50}$$

First term in the Hamiltonian $\hat{H}$ in equation (49) represents the kinetic energy of the moving electron whereas second term $V(\vec{r})$ represent the external position dependent potential. This potential is generated mainly by the different atoms of the system. The last term accounts for the

interaction between different electrons in the system. The electron-electron interaction energy operator $U(\vec{r}_i, \vec{r}_j)$ and kinetic energy operator are universal operators whereas external potential energy operator $V(\vec{r})$ is system dependent. In order to obtain the ground state energy of the system, one has to solve the Schrödinger equation given in the equation (50). But due to the many particle interaction term $U(\vec{r}_i, \vec{r}_j)$, the Hamiltonians is not separable and direct solution is highly computationally expensive. The presence of the last term makes finding solution for this system a formidable task.

Section B.01 Kohan and Sham Density function theory (DFT)

In 1965, Hohenberg Kohan and Sham have proposed an elegant method in order to overcome this difficulty (Kohn and Sham 1965, Hohenberg and Kohn 1964). In their proposal, all electron interaction had been replaced by an effective nonlocal potential. Furthermore they have proven that the total energy of the system is completely determined by the electron density. This has an important consequence; if one minimizes the total energy function of the system, the minimum value of the total energy is exactly equal to the ground state energy of the system (Jones and Gunnarsson 1989). In other words the density at which, the total energy minimize is exactly equal to the ground state particle density. The total energy of many particle systems can be represented by equation (51);

$$E[\{\psi_i\}] = 2\sum_i \int \psi_i \left[-\frac{\hbar^2}{2m}\right] \nabla^2 \psi_i d^3r + \int V_{ion}(r) n(r) d^3r + \frac{e^2}{2} \int \frac{n(r)n(r')}{|r-r'|} d^3r d^3r' + E_{XC}[n(r)] + E_{ion}(\{R_I\}) \quad (51)$$

Where m is the mass of the electron and e is the charge. In the above representation the electron density is given by;

$$n(r) = 2\sum_{i} |\psi_i(r)|^2 \tag{52}$$

This formulation produced set of n equations where n is equal to the number of particles in the system. Each particle is then subject to effective potential due to all other particles. In order to obtain ground state of a system, one must solve Schrödinger equations (53) self consistently. In this scheme one should first solve the equation (53) for a given initial set of wave functions. Then the resulted charge density yields new set of potentials, as potentials are function of particle density. New equation then again solve for wave functions. This process is repeated until a predetermined tolerance criteria is met.

$$\left[\frac{-\hbar^2}{2m}\nabla^2 + V_{ion}(r) + V_H(r) + V_{XC}(r)\right]\psi_i(r) = \varepsilon_i\psi_i(r) \tag{53}$$

In this equation $V_{\mathrm{H}}(\mathrm{r})$ is called Hatree potential (Almbladh and et al. 1983), which describes the interaction between two electrons at points in space r and r'.

$$V_H(r) = e^2\int \frac{n(r')}{|r-r'|}d^3r' \tag{54}$$

The $V_{\mathrm{XC}}$ represents the exchange correlation potential .

$$V_{XC}(r) = \frac{\delta E_{XC}[n(r)]}{\delta n(r)} \tag{55}$$

This exchange correlation energy function, $E_{XC,}$ in equation (55) is not known exactly (Uday and Rajagopal 1980, MacDonald and et al. 1980, Almbladh and Pedroza 1984). Hence it is necessary to utilize an approximate method in generating starting exchange correlation energies.

Section B.02 Local density approximation (LDA)

One method to generate exchange correlation energy is to assume that exchange correlation energy at a given point r is that of homogeneous electronic gas with same density at point r (Wigner 1938, Kohn and Sham 1965, Hedin and Lundqvist 1971, Vosko and et al. 1980). This approximation, first prediction by Hohenberg Kohan, is the most widely used simplest assumption known as local density approximation (LDA) (Kohn and Sham 1965, Langreth and et al. 1981, Langreth and et al. 1983). This is given by equation (A.8).

$$E_{XC}[n(r)] = \int \varepsilon_{XC}(r)\, n(r) d^3r \tag{56}$$

$$\frac{\delta E_{XC}[n(r)]}{\delta n(r)} = \frac{\partial [n(r)\varepsilon_{XC}(r)]}{\partial n(r)} \tag{57}$$

With LDA $\varepsilon_{XC}$ is approximated with its homogeneous counterpart as;

$$\varepsilon_{XC}(r) = \varepsilon_{XC}^{hom}[n(r)] \tag{58}$$

Even with these implications obtaining a solution for Kohan Sharm equation remains a formidable task. The main difficulties remain with the choice of suitable set of wave functions.

Section B.03 Crystal lattice and Bloch Theorem

The easy and obvious first choice is to use plane waves as a basis functions. But this choice introduces an additional difficulty. In order to describe an infinite number of atoms in continuous space requires infinite number of wave functions. This difficulty is then overcome by the use of Bloch theorem (Bloch 1929). Bloch theorem state that the properties of a wave function for a periodic lattice can be captured into a product of cell-periodic part, which captures all essential properties of the lattice, and continuous wave like part (Stoddart and Hanks 1978, Hanks and Stoddart 1979). Then the continues basis set can be substituted by a discrete set of plane wave basis set whose wave vectors are equal to the reciprocal lattice vector of the lattice. Resultant wave function can be written as in equation (59):

$$\psi_i(r) = \sum_G c_{i,k+G}\ exp[i(k+G)\cdot r] \tag{59}$$

The use of plane wave with the Bloch theorem requires the periodicity. But for a molecule, such as a DNA base pair, it is difficult to define a periodic lattice. Continuous basis set is then required to deal with such systems. This difficulty can be somewhat overcome by using a large super cell which enclose the molecule and sufficiently large vacuum (Gollisch and Fritsche 1981). This approach assumes that the super cell (Norman 1986, Mohn and Schwarz 1993, Krompiewski 1998) is so large that molecule in the neighboring cell does not feel each other.

Section B.04 Beyond plane waves: Basis functions

Even with such methods, the treatment of molecules remains questioning. More elegant way of dealing this is to use a basis set which closely resembles the atoms present in the system. For example Hydrogen atom only has a one "s" orbital whereas carbon atom has "s" and "p" orbital. On the other hand, if one only treats valence electrons in the computation then all first and second row elements share somewhat same configuration (Davidson and Feller 1986). Then one can define a set of basis function for each row in the periodic table. The advantage of such treatment is that the problem could be easily converged to the desired state as it now resembles the properties of the system.

$$STO = \frac{\zeta^3}{\sqrt{\pi}} e^{(-\zeta r)} \tag{60}$$

The most primitive type used is the Slater type functions (Slater 1930, Hehre, Stewart and Pople 1969) . These functions correspond to a set of functions which decayed exponentially with distance from the nuclei. The computational cost associated with Slater type functions is tremendously high.

$$GTO = \frac{2\chi}{\pi^{0.75}} e^{(-\chi r^2)} \tag{61}$$

This difficulty comes with the monotonic dependence in r in the exponent. By using Gaussian type functions, where r has quadratic dependence in the exponent, computational cost can be reduced significantly, as now product of two Gaussian functions again produces a Gaussian function. The price we have to pay is the accuracy. But due to the computational ease, at present, Gaussian type functions (Hehre et al. 1969, Newton et al. 1969) are excessively in use. Slater functions could be approximated by the linear combination of Gaussian functions. By adding more exponents to the basis function it is possible to approximate Slater type closely.

## Section B.05 Minimum basis set

The minimum basis set composes of a minimum number of basis function required to perform all atom computation. Commonly the basis set, STO-nG is used as a minimum basis set for molecules containing lighter atoms. (Newton et al. 1969) Where n represents number of Gaussian primitives use to construct the basis functions. Often minimum basis set is not sufficient for accurate results. The accuracy of the results can be increased by accurately selecting the basis set. There are two possible improvements. polarization functions (Hariharan and Pople 1973) and diffuse functions could be added to the valence orbital basis functions. This could add an additional flexibility to the basis set and it will help the accuracy of estimation of the complex electron distribution in a molecule. As an example, in the minimal basis set hydrogen atom only contains s orbital. But when hydrogen atoms make bonds with some other atoms, the bond may not resemble correctly by only having s orbital due to lack of flexibility of the basis set. By adding an additional p orbital to the basis set of hydrogen atom would provide the needed flexibility and it would help to produce accurate bonding characterization. Another weakness of the minimum basis set is that it does not accurately represent the slowly decaying tail portion of the atomic orbital. This can be overcome by adding a shallow Gaussian orbital to the basis set. A * sign is used to indicate that polarization functions are added to the basis set

whereas a + sign is used to indicate that diffuse functions are added to the basis set. And additional * or + indicate that the light atoms also contain polarization function and diffuse functions respectively.

Section B.06 Basis set selection

Most commonly used basis set compose of more than one function to describe the valence electrons of the system. These basis sets are called split-valence basis sets commonly has the X-YZg format. X represents the number of Gaussian primitives in each core basis functions. The Y and Z indicate that each valence orbital composes of two basis functions and first function has Y number of Gaussian primitives and second function has Z number of Gaussian primitives. As an example the most commonly use basis function in DNA computations ,6-31G* (Davidson and Feller 1986, Moran et al. 2006) has six primitive Gaussians for each core orbital and all its valence orbital are represent by two sets of functions. First function has 3 Gaussian primitives and the other has only one Gaussian primitive. Move over a polarization functions is also present in this basis set.

As an example listing (1) illustrate the output of EMSL Basis Set Exchange Library(Schuchardt et al. 2007) for carbon atom.

```
! 3-21G  EMSL  Basis Set Exchange Library   9/8/10 1:44 PM
! Elements                      References
! --------                     ----------
!  H - Ne: J.S. Binkley, J.A. Pople, W.J. Hehre, J. Am. Chem. Soc 102 939 (1980)
! Na - Ar: M.S. Gordon, J.S. Binkley, J.A. Pople, W.J. Pietro and W.J. Hehre,
!        J. Am. Chem. Soc. 104, 2797 (1983).
!  K - Ca: K.D. Dobbs, W.J. Hehre, J. Comput. Chem. 7, 359 (1986).
! Ga - Kr: K.D. Dobbs, W.J. Hehre, J. Comput. Chem. 7, 359 (1986).
! Sc - Zn: K.D. Dobbs, W.J. Hehre, J. Comput. Chem. 8, 861 (1987).
!  Y - Cd: K.D. Dobbs, W.J. Hehre, J. Comput. Chem. 8, 880 (1987).
! Cs     : A 3-21G quality set derived from the Huzinage MIDI basis sets.
!        E.D. Glendening and D. Feller, J. Phys. Chem. 99, 3060 (1995)
!
```

```
****
C     0
S   3   1.00
    172.2560000          0.0617669
     25.9109000          0.3587940
      5.5333500          0.7007130
SP   2   1.00
      3.6649800         -0.3958970          0.2364600
      0.7705450          1.2158400          0.8606190
SP   1   1.00
      0.1958570          1.0000000          1.0000000
****
```

Since these parameters is for 3-21G basis function, and C atom has electrons in 1s, 2s, 2p orbital; there will be three Gaussian determinants for core orbital (i.e. 1S) and the valence orbital 2S and 2P should have two.

Table 6 : 3-21 G basis set for C

| Atom | $\propto_{1si}$ | $d_{1si}$ | $\propto_{2si}=\propto_{2pi}$ | $d_{2si}$ | $d_{2pi}$ | $\propto'_{2si}=\propto'_{2pi}$ |
|---|---|---|---|---|---|---|
| C | 172.256 | 0.0617669 | 3.66498 | -0.395897 | 0.236400 | 0.195857 |
| | 25.9109 | 0.3587940 | 0.770545 | 1.2158400 | 0.860619 | |
| | 5.53335 | 0.7007130 | | | | |

$$\Phi_{1s}(r) = \sum_{i=1}^{3} d_{1si}\Phi_{1s}^{GF}(r, \propto_{1si})$$

$$= 0.617\Phi_{1s}^{GF}(r, 172.256) + 0.3587\Phi_{1s}^{GF}(r, 25.910)$$

$$+ 0.7007\Phi_{1s}^{GF}(r, 5.533) \qquad (62)$$

$$\Phi_{2s}(r) = \sum_{i=1}^{2} d_{2si}\Phi_{2s}^{GF}(r, \propto_{2si}) + d'_{2s}\Phi_{2s}^{GF}(r, \propto'_{2s})$$

$$= -0.395\Phi_{2s}^{GF}(r, 3.664) + 1.215\Phi_{2s}^{GF}(r, 0.771)$$

$$+ 1.000\Phi_{2s}^{GF}(r, 0.195) \qquad (63)$$

$$\Phi_{2p}(r) = \sum_{i=1}^{2} d_{2pi}\Phi_{2p}^{GF}\left(r, \propto_{2pi}\right) + d'_{2p}\Phi_{2p}^{GF}\left(r, \propto'_{2p}\right)$$

$$= 0.236\Phi_{2p}^{GF}(r, 3.664) + 0.860\Phi_{2p}^{GF}(r, 0.771)$$

$$+ 1.000\Phi_{2p}^{GF}(r, 0.195) \qquad (64)$$

These functions then use as the starting point for computer molecular orbital computations.

Appendix C. Parameters for tight binding formulism

Onsite energy and transfer integrals are usually calculated using first principle methods (Appendix B). In any first principle methods the first step is to select appropriate a basic set which specify the atomic orbital for all atoms present in the system. In the case of charge transfer properties in the DNA, where charge transfer integrals play a critical role, the natural selection is to use split valence triple zeta basis set, e.g. 6-31G++ or TZ2P, which describes the trail of the wave functions accurately. The tail of the wave function mainly determines the charge transfer integral as charge transfer integral depends on the effective overlap between two wave functions. In general the first step of such procedure is to obtain the optimal structure by total energy minimization methods describe in the Appendix C . The optimized atomic orbital (Porezag et al. 1995) produced by this procedure is then serves as the basis functions for the transfer integral and onsite energy calculations.

Section C.01 Fragmental Molecular orbital approach

As an example, the molecular orbital for the DNA base can be written as a linear combination of the optimized atomic orbital $\eta_\mu$.

$$\phi_i = \sum_\mu c_\mu^i \eta_\mu \tag{65}$$

Where summation μ is over total number of atoms in the base and $c_\mu$ is computed using first principle optimization. In this framework on-site energy of the base is given by the matrix element $T_{ii}=\varepsilon_{ii;}$

$$\epsilon_i = -\langle \phi_i | \widehat{H}_{KS} | \phi_i \rangle \tag{66}$$

The table 1 illustrates the ionization energies for the isolated base pairs. The absolute values of these energies depend on the computation methods employed. These energies were calculated,

for optimized isolated base pairs, using Amsterdam Density Function (ADF) software package with TZ2p split valence basis set. The cutoff distance of 10Å was used to select atoms participate in the calculations. We have computed the ionization energies described in Table 5 using dry DNA bases. This was done in order to mimic the experimental situations where the I-V characteristic is measured on the dry DNA. However there is a debate among the scientist about how dry is the dry DNA is. It is commonly believed that even in the DNA which commonly considered as dry, consists of some water molecules bonded to it which never leaves. We haven't considered this situation as it is not clear the extent of such hydration. Our calculation are performed in isolated DNA fragments, i.e. G or A.

Section C.02 Transfer integral Computation

The transfer integral between two molecular orbital $\phi_i$ and $\phi_j$ is given by the matrix element $T_{ij}$ as;

$$T_{ij} = \langle \phi_i | \widehat{H}_{KS} | \phi_j \rangle \tag{67}$$

In order to accurately represent charge transfer through molecular fragments, the molecular orbital $\phi_i$ needs to be orthogonal to each other. But in general, computations are carried out with non-orthogonal fragmental molecular orbital basis sets (Mehrez and Anantram 2005). The correct charge transfer integral is then computed as;

$$T_{ij}{}' = T_{ij} - S_{ij}(T_{ii} + T_{jj})/2 \tag{68}$$

Where $S_{ij}$ is the overlap matrix in the molecular basis set is given by;

$$S_{ij} = \sum_{\mu\nu} c_\mu^i \, c_\nu^j \langle \eta_\mu | \eta_\nu \rangle = \sum_{\mu\nu} c_\mu^i \, c_\nu^j S_{\mu\nu} \tag{69}$$

In the above equation $S_{\mu\nu}$ is the overlap matrix in the atomic basis set. $T_{ij}$ is the transfer matrix element in non orthogonal basis set given by;

$$T_{ij} = \sum_{\mu\nu} c_{\mu}^{i}\, c_{\nu}^{j} \langle \eta_{\mu} | \widehat{H}_{KS} | \eta_{\nu} \rangle = \sum_{\mu\nu} c_{\mu}^{i}\, c_{\nu}^{j} H_{\mu\nu} \tag{70}$$

$H_{\mu\nu}$ is the Hamiltonian the atomic basis set (Kubar□ et al. 2008).

## Appendix D. Parallel and Distributed Computing

Even though processing power of modern computers are significantly higher than that of a decade ago computers the trend is flattening out. Moreover manufactures has started to integrate more and more processing units (cores) into a central processing units (CPU) rather than trying to develop faster single core CPU. Though this has obvious advantage over power consumption, harnessing the collective computational power is challenging. In this prospective parallel and distributed high performance computing is beginning to play an important part in future science.

The main objective of a computational scientist is to process input data in a meaningfull way producing new information about a given problem. This question then can be translated into three main steps; feeding data into the main processing unit, processing data and writing new information out. The program performance depends on the way these three sections perform.

(a) Inputs

Inputs can be fed into the program by many means. It can be read from the file in the hard disk each time it needed, or it can be read from the hard disk once and store in a variable in a memory location. Though these two mechanisms serve the same purpose, performancewise it has immense difference. Typically if data reside in the hard disk the maximum attainable data transfer rate is roughly 70MB/s whereas if data reside in the shared memory, with Intel xeon CPU, up to 32 GB/s data transfer rate can be achieved. If data resided in a CPU cash memory then computation can access to those memory faster than if it is in the main memory more over if it is in a register then CPU can access it even faster, within few flop delay. The careful management of data flow would change program run time from days to hours for large data intensive computations.

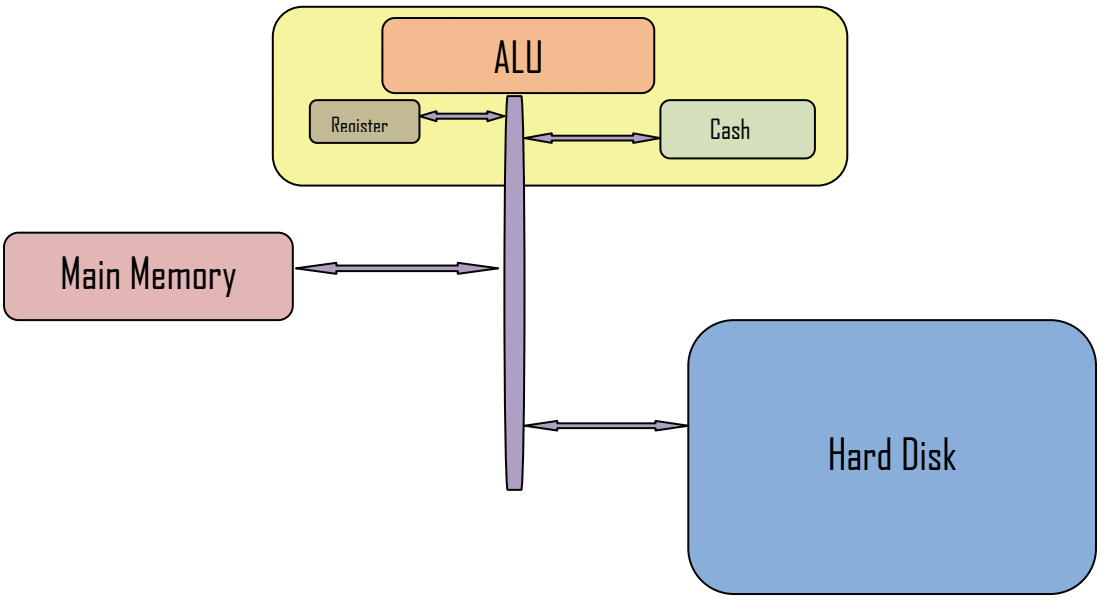


Figure 41 : Schematic illustration of a system architecture of a computer memory hierarchy

Modern computer architecher had been designed to support large data management requirements. For this reason the data transfer between main memory and the cash is performed in data chunks rather than individual elements. Hence when a program requests for a data from memory, it usually receives a chunk of data from the memory. Since the data transfer in an expensive operation it is important to design the program data structure in such a way that once a data request if fulfill program perform as much as computations on current data in the cash before requesting another data from main memory.

(b) Data layout

The present computational problem the input data are wave function at each site (its real and imaginary part), lattice displacement and twist angle, first derivative of lattice displacement and twist angle. Since the Hamiltonian operates on only nearest neighbors it is fruitful to use

contiguous data allocation. The continuous memory space of main memory can be allocated in C language using “malloc” command.

*ptr=(cast-type*)malloc(byte-size);*

This command dynamically allocates a continuous memory space of size given in bytes and returns a reference to the memory location.

The Figure 42 shows two possible data layouts. Both one dimensional layout consist of same number of data and memory requirement is also same. In the first situation data is arranged as structure of arrays whereas in the second format data is arranged as array of structures. The main difference of these two situations is when the program asks for $Y_1$ (y stretching) data element, computer memory management system delivers first few elements of the array (i.e. $Y_2$, $Y_3$ …). But in order to compute the next time step of $Y_1$ it also needs the value of $\Theta_1$ (twist angle), $\Omega_1$ (angular acceleration) etc. Owing to the fact CPU only has limited cash memory, each time program requires new data element (say $\Theta_1$ ) it will wipe out the rest of previous data resides in the cash (i.e. $Y_2$, $Y_3$ …). Since memory operations are relatively costly operations, in themes of flops, it is preferable to use second data layout. In this method data is stored in a structured fashion, so that the data request for data element $Y_1$ brings almost the entire required elements into the cash. Now the processor can perform computation without waiting for data.

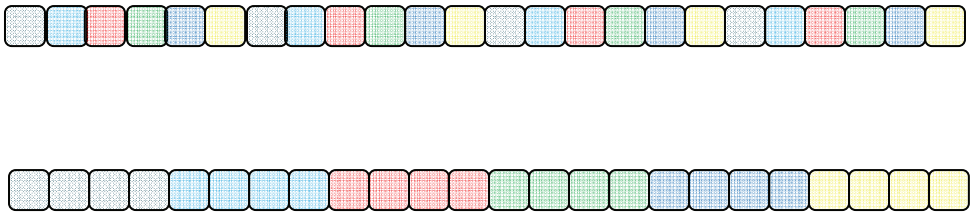

Figure 42 : Array of structures Vs Structure of arrays.

Listing 1 demonstrates the declaration of an array of structures which is used in the program.

```
typedef struct
        {
        double HR;
        double HI;
                double XR;
        double XI;
                double TH;
                double OM;
        double Y;
        double A;

         } Base_Pair;

Base_Pair *BP;
BP=  malloc (N*sizeof(Base_Pair));
```

(c) Parallel Processing

In order to speed up the computation, many processing cores have been used in the simulations. But this introduced additional overhead. The computations are carried out at a cluster where each processing unit (node) equipped with its own hardware including its own memory space. A particular node only has exclusive access to its own memory space. This requires distribution of data trough out the processing nodes prior to any computation and

gathering results back after computations. Massage Passing interface (MPI) was used to communicate between nodes.

MPI can be initiated using MPI_Init commands. This command takes all arguments which are used to initiate the main process and then initial addition number of process requested by the user with same arguments as with main program. Each process is then assigned as identification number from zero to number of process and communication reference point (communicator) is provided.

*int MPI_Init(int *argc, char ***argv)*

Input data is then distributed throughout the processing units. It is important to balance computational load throughout all processing node since if one node becomes over loaded with large number of data then it will become a bottleneck and all other processors has to wait till overloaded process to finish. Distributing equal number of data elements to each node is an important task in terms of performance. Initial data distribution was carried out by MPI_Scattre command which as following syntax. Since MPI_Scatter is a collective operation each process in the communicator should call this simultaneously.

```
int MPI_Scatter (
      void *sendbuf,
      int sendcnt,
      MPI_Datatype sendtype,
      void *recvbuf,
      int recvcnt,
      MPI_Datatype recvtype,
      int root,
      MPI_Comm comm )
```

In this command first argument sendbuf refers to the pointer to the input data, second and third argument refer to the number of data to be sent to each node and the type of data is to be sent.

Next three arguments refer to the pointer to the receiving variable, number of elements to be received, and the type of data to be expected respectively. Seventh argument specify the node which is initiating the scattering process and the last argument is the communicator generated by the MPI_Init() command.

As illustrated by listing 1 in order to compute a particular $Y_n$ $\Theta_n$ $\Psi_n$ values program need its neighboring values also. When input data is distributed across different nodes, the computation of edge element brings additional problems hence now the neighboring elements are resided in different node. So at each calculation steps these values should move to the computation node. Point to point communication is performed using MPI_Send() and MPI_Recv() commands.

*int MPI_Send( void *buf, int count, MPI_Datatype datatype, int dest,int tag, MPI_Comm comm )*
*int MPI_Recv( void *buf, int count, MPI_Datatype datatype, int source,int tag, MPI_Comm comm, MPI_Status *status )*

Since above commands are blocking commands the possibility of getting into a deadlock situation is significantly high. In order to avoid such situation I have used the combine version of the send receive command.

*int MPI_Sendrecv( void *sendbuf, int sendcount, MPI_Datatype sendtype,int dest, int sendtag,void *recvbuf, int recvcount, MPI_Datatype recvtype,int source, int recvtag, MPI_Comm comm, MPI_Status *status )*

In this command first five arguments describe the behavior of outgoing data, whereas the next five describe the incoming data. Last argument returns the status of the receiving process. Each process has a rank in the communicator. The initial data distribution was done acceding to the hierarchy of the processes. Then each process need to send its edge data to the processors who's rank is one less or greater than its own. Following code segment illustrate the data distribution.

```
MPI_Comm_size ( MPI_COMM_WORLD, &size );
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
M=N/size;
Send_count_start=M;
        Destination=myrank+1;
        if (Destination>size-1)
        {
                Destination=0;
        }
        Origin=myrank-1;
        if (Origin<0)
        {
                 Origin=size-1;
        }
MPI_Sendrecv( &BP[Send_count_start], ghost,MPI_DOUBLE,Destination,4,&BP[0],
ghost,MPI_DOUBLE, Origin,4,MPI_COMM_WORLD, &M_status);
        Send_count_start=ghost;
        if (Send_count_start>N)
        {
                Send_count_start=0;
        }
        Destination=myrank-1;
        if (Destination<0)
        {
                Destination=size-1;
        }
        Origin=myrank+1;
        if (Origin>size-1)
        {
                 Origin=0;       }

MPI_Sendrecv(&BP[Send_count_start],ghost,MPI_DOUBLE,Destination,3, &BP[M+ghost],
ghost,MPI_DOUBLE, Origin,3,MPI_COMM_WORLD, &M_status);
```

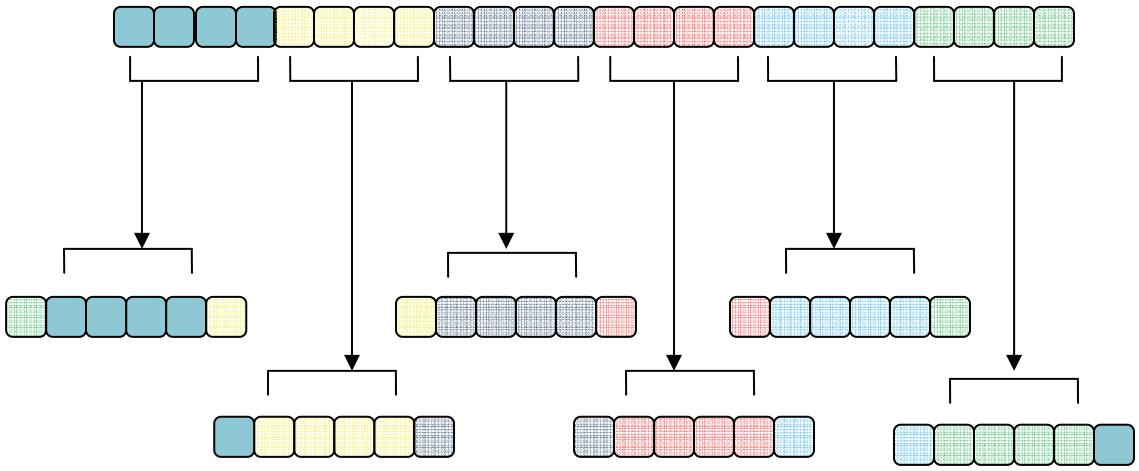

Figure 43 : Data distribution pattern for MPI parallel programming.

Finally escape was determined as the charge density of donor site reach the threshold value. For this purpose previously scattered data need to be gathered into one location. This process is achieved by the MPI_Gather() command.

*int MPI_Gather ( void *sendbuf, int sendcnt, MPI_Datatype sendtype,void *recvbuf, int recvcount, MPI_Datatype recvtype,*
*int root, MPI_Comm comm )*

(d) Output

In order to avoid race condition, where many processes try to write into same location, an output oriented algorithm was used. Since it is possible to read same input by several processes without any apparent consequence the output oriented algorithm is preferred.

(e) Performance considerations

The speedup gain through parallel execution is depends on both computation time and communication time as in parallel programs communication serves as an overhead compares to serial version.

$$Speedup = \frac{Sequential\ execution\ time}{Parallel\ execution\ time} \tag{71}$$

$$Efficiency = \frac{Sequential\ execution\ time}{Processors\ used \times Parallel\ execution\ time} \tag{72}$$

As given by the Amdahl's law the denominate factor of the speed up in a parallel algorithm is governs by the serial portion of the algorithm. Simple arithmetic justifies that even for a code which is 90% runs in parallel mode the maximum achievable speed up is only 4.7. Due to this reasons for small base pair sequences the simulation was executed in batch mode. In which serial code was executed on different nodes with different initial conditions. But as DNA strand gets larger parallel algorithm was used with 4 nodes per run. This allows keeping communication overhead under control while achieving additional advantage due to small dataset which facilitate efficient use of cash memory.

## Appendix E. Software

### (a) Charge transport through DNA

#### *(i) Main Program (C/C++ MPI)*

```
/******************************************************************************/
/*
  Purpose:
    Structural Fluctuation and Radiation Damage to DNA
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
   16 Aug 2010
  Parallel Version: C MPI
  Author:
Neranjan Edirisinghe
*/
/******************************************************************************/

# include "stochastic_rk.h"


double fi ( double x );
double gi ( double x );
/******************************************************************************/

int main(argc , argv)
int argc;  /* contains number of arguments */
char **argv; /* contains the arguments themselves */
{//{1}
/******************************************************************************/

/* variable definitoion*/
        Int
Position,n,M,size,myrank,QD,i,j,len,Energy_Min,N_Index,Red_Count,Send_count_start,Syncronization_Interval,Exi
t_Switch,Buffer_Size,Local_Index;
        double  *E,*THO,X,QW1,THO1,V,h,E1,E2,*Total_Energy,q=1;
        double
count,t,Kb,gma,m,Tresh_hold,Tresh_hold_Evoluation,Energy_Tol,Norm1,radiation_rate,Relax_Time,Hydration_En
ergy;
   double t0 = 0.0;
   double tn = 1;
   int
Radiation_Location,seed,T,Print_Interval,Pre_Escape2,Pre_Escape1,Energy_Correction_Counter,Parallel_debug,
EXIT_ON,Buffer_Index;
        int
compleate,Escaped,Restart_Switch,Radiation_Study,Transport_Study,Energy_Correction_Interval,Radiation_Hit,P
rinting_On;
        int Buffer_Print_Interval,Buffer_Print_Advance;
        double Elaped_time,Parati;
        char *Initial;
        char Prefix[250]={0};
        char OutPut[250]={0};
        char *Restart_File;
```

```
        char *OutPut_File;
        char output_file [250];
        char Index[250] ={0};
        float CH;
        struct stat buffer;
        int
status,Destination,Origin,Job_ID,Initial_Given,Initial_Temperature,Final_Temperature,Temperature_Interval;

        Norm Exit_Conditions;
        MPI_Status M_status;
                Base_Pair *BP,*y1,*y2,*y0,*ytemp;
                Base_Pair *BP_MAIN;
                Radiation Ray;
                Base *Strand;
                Storage_Base_Pair *Output_Buffer;
                Storage_Base_Pair *Master_Output_Buffer;
                Cell cell_acceptor;
                Cell cell_donor;
                Base base_acceptor;
                Base base_donor;
                div_t nh;


/*****************************************************************************/
/*
Open file for write output
*/
/*****************************************************************************/
        FILE *pfile;
        FILE *Restart;
        FILE *IN;
        FILE *Data;
        FILE *yfile;
        FILE *afile;
        FILE *tfile;
        FILE *ofile;
        FILE *wfile;
        FILE *hfile;
        FILE *tmfile;


/*****************************************************************************/
/*
Define base pair structure
*/
/*****************************************************************************/
                BP_MAIN=  malloc (N*sizeof(Base_Pair));
                if (BP_MAIN == NULL)
                {//{2}
                        printf("error defining BP_MAIN  ");
                        exit(EXIT_FAILURE);
                }//{2}
```

```
		BP=  malloc (N*sizeof(Base_Pair));
		if (BP == NULL)
		{//{2}
			printf("error defining BP  ");
			exit(EXIT_FAILURE);
		}//{2}


/*************************************************************************/
		/*************************************************************************************/
/*
Define E variable
*/
/*************************************************************************/
		E= malloc (N*sizeof(double));
		if (E == NULL)
		{//{2}
			printf("error defining E  ");
			exit(EXIT_FAILURE);
		}//{2}
/*************************************************************************/
/*
Define THO variable
*/
/*************************************************************************/
		THO=malloc (N*sizeof(double));
		if (THO == NULL)
		{//{2}
			printf("error defining THO  ");
			exit(EXIT_FAILURE);
		}//{2}
/*************************************************************************/
		//IN=fopen(argv[1],"r");
MPI_Init(&argc, &argv);
MPI_Comm_size ( MPI_COMM_WORLD, &size );
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
M=N/size;
		Strand=  malloc ((M+2*ghost)*sizeof(Base));
		if (Strand == NULL)
		{//{2}
			printf("error defining Strand  ");
			exit(EXIT_FAILURE);
		}//{2}

		Total_Energy=  malloc ((M+2*ghost)*sizeof(double));
		if (Strand == NULL)
		{//{2}
			printf("error defining Strand  ");
			exit(EXIT_FAILURE);
		}//{2}

		y1=malloc ((M+2*ghost)*sizeof(Base_Pair));
		if (y1 == NULL)
		{
			printf("error defining d1  ");
```

```
                exit(EXIT_FAILURE);
        }

        y2=malloc ((M+2*ghost)*sizeof(Base_Pair));
        if (y2 == NULL)
        {
                printf("error defining d2  ");
                exit(EXIT_FAILURE);
        }

        y0=malloc ((M+2*ghost)*sizeof(Base_Pair));
        if (y0 == NULL)
        {
                printf("error defining d2  ");
                exit(EXIT_FAILURE);
        }



        ytemp=malloc ((M+2*ghost)*sizeof(Base_Pair));
        if (ytemp == NULL)
        {
                printf("error defining dtemp  ");
                exit(EXIT_FAILURE);
        }

if (myrank==0)
{///{2}
        if (DEBUG==0)
        {///{3}
                Data=fopen(argv[3],"r");
        }///{3}
        else
        {///{3}
                Data=fopen("data.txt","r");
        }///{3}
                fscanf (Data, "%f", &CH);
                X=CH;
                fscanf (Data, "%f", &CH);
                Tresh_hold=CH;
                fscanf (Data, "%f", &CH);
                Tresh_hold_Evoluation=CH;
                fscanf (Data, "%d", &QD);
                fscanf (Data, "%d", &Print_Interval);
                fscanf (Data, "%d", &Restart_Switch);
                fscanf (Data, "%f", &CH);
                radiation_rate=CH;
                fscanf (Data, "%f", &CH);
                Relax_Time=CH;
                fscanf (Data, "%d", &Red_Count);
                fscanf (Data, "%d",&Radiation_Study);
                fscanf (Data, "%d",&Transport_Study);
                fscanf (Data, "%d",&Energy_Correction_Interval);
                fscanf (Data, "%d",&Syncronization_Interval);
                fscanf (Data, "%d",&Parallel_debug);
                fscanf (Data, "%d",&T);
```

```
                fscanf (Data, "%d",&Initial_Given);
                fscanf (Data, "%d",&Initial_Temperature);
                fscanf (Data, "%d",&Final_Temperature);
                fscanf (Data, "%d",&Temperature_Interval);
                fscanf (Data, "%d",&Printing_On);
                fscanf (Data, "%d",&Buffer_Size);
                fclose(Data);
                printf ("    Restart            :  ");
                printf ( "%d\n    ", Restart_Switch);

}//{2}
/*****************************************************************************/
MPI_Barrier(MPI_COMM_WORLD);
if (Parallel_debug)
{//{2}
        printf(" I have reached BC START point %d \n",myrank);
}//{2}

MPI_Bcast(&X,1,MPI_DOUBLE,0,MPI_COMM_WORLD);
MPI_Bcast(&Tresh_hold,1,MPI_DOUBLE,0,MPI_COMM_WORLD);
MPI_Bcast(&Print_Interval,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Restart_Switch,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Red_Count,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Relax_Time,1,MPI_DOUBLE,0,MPI_COMM_WORLD);
MPI_Bcast(&radiation_rate,1,MPI_DOUBLE,0,MPI_COMM_WORLD);
MPI_Bcast(&Radiation_Study,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Transport_Study,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Energy_Correction_Interval,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&QD,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Syncronization_Interval,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Parallel_debug,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&T,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Initial_Given,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Initial_Temperature,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Final_Temperature,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Temperature_Interval,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Printing_On,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Buffer_Size,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Bcast(&Tresh_hold_Evoluation,1,MPI_DOUBLE,0,MPI_COMM_WORLD);
if (Parallel_debug)
{//{2}
printf(" I have reached BC point %d \n",myrank);
}//{2}
/*****************************************************************************/

                Output_Buffer=  malloc (M*Buffer_Size*sizeof(Storage_Base_Pair));
                if (Output_Buffer == NULL)
                {//{2}
                        printf("error defining Output_Buffer  ");
                        exit(EXIT_FAILURE);
                }//{2}


                Master_Output_Buffer=  malloc (N*Buffer_Size*sizeof(Storage_Base_Pair));
                if (Master_Output_Buffer == NULL)
                {//{2}
```

```
                        printf("error defining Master_Output_Buffer  ");
                        exit(EXIT_FAILURE);
                }///{2}


Hydration_Energy=12.27;
        //T=0;
        compleate=1;
        if (DEBUG==0)
        {///{2}
                Initial=argv[1];
        }///{2}
        else
        {///{2}
                Initial="L.1";
        }///{2}

        len=strlen(argv[1]);
        strncpy(Index,Initial+2,len-2);
        N_Index=atoi(Index);

        Job_ID=atoi(argv[4]);
        //if (Parallel_debug)
        //{///{2}
                printf("%d \n",N_Index);
        //}
        seed = pow(N_Index,2);
        printf("%d \n",seed);
        /*****************************************************************************/
PutSeed(seed);

while (compleate)
{///{2}
printf("done\n %d %d \n",T,myrank);
Elaped_time=0;
Elaped_time=-MPI_Wtime();



Ray.Sub_unit=0;
Ray.Position=0;
Ray.Ray_Hit=0;
cell_acceptor.Initiate=0;
cell_acceptor.Sub_unit=0;
cell_acceptor.Position=0;
base_acceptor.well_depth=0;
base_acceptor.absorption=0;
cell_donor.Initiate=0;
cell_donor.Sub_unit=0;
cell_donor.Position=0;
base_donor.well_depth=0;
base_donor.absorption=0;
Pre_Escape2=Pre_Escape1=1;

/*****************************************************************************/
// Initialte Variable BP
```

```
/*****************************************************************************/
        for (j=0;j<N;j++)
        {//{3}
                BP[j].HR=0.0;
                BP[j].XR=0.0;
                BP[j].HI=0.0;
                BP[j].XI=0.0;
                BP[j].Y=0.0;
                BP[j].A=0.0;
                BP[j].OM=0.0;
                BP[j].TH=0.0;

                BP_MAIN[j].HR=0.0;
                BP_MAIN[j].XR=0.0;
                BP_MAIN[j].HI=0.0;
                BP_MAIN[j].XI=0.0;
                BP_MAIN[j].Y=0.0;
                BP_MAIN[j].A=0.0;
                BP_MAIN[j].OM=0.0;
                BP_MAIN[j].TH=0.0;



        }//{3}


        for (j=0;j<M+2*ghost;j++)
        {//{3}
                Strand[j].well_depth=0;
                Strand[j].absorption=0;
                Total_Energy[j]=0;
                y1[j].HR=0.0;
                y1[j].XR=0.0;
                y1[j].HI=0.0;
                y1[j].XI=0.0;
                y1[j].Y=0.0;
                y1[j].A=0.0;
                y1[j].OM=0.0;
                y1[j].TH=0.0;
                y2[j].HR=0.0;
                y2[j].XR=0.0;
                y2[j].HI=0.0;
                y2[j].XI=0.0;
                y2[j].Y=0.0;
                y2[j].A=0.0;
                y2[j].OM=0.0;
                y2[j].TH=0.0;
                ytemp[j].HR=0.0;
                ytemp[j].XR=0.0;
                ytemp[j].HI=0.0;
                ytemp[j].XI=0.0;
                ytemp[j].Y=0.0;
                ytemp[j].A=0.0;
                ytemp[j].OM=0.0;
                ytemp[j].TH=0.0;
```

```
}///{3}
for (j=0;j<N;j++)
{///{3}
        E[j]=0;
        THO[j]=0;
}///{3}




if (myrank==0)
                {///{3}
                        if (!Restart_Switch)
                        {///{4}

/*************************************************************************/
                /*
                Creat Input Files
                */

/*************************************************************************/
                                if (Initial_Given)
                                {///{5}

                                                if (DEBUG==0)
                                                {///{6}
                                                        Initial=argv[2];
                                                }///{6}
                                                else
                                                {///{6}
                                                        Initial="L.1";
                                                }///{6}

                                        sprintf(output_file, "%d", T);
                                        strcat(OutPut,Initial);
                                        strcat(OutPut,"_Displacement_Initial_");
                                        strcat(OutPut,output_file);
                                        strcat(OutPut,".txt");
                                        //printf("%s\n",OutPut);
                                        OutPut_File=OutPut;
                                        yfile=fopen(OutPut_File,"r");
                                        len=strlen(OutPut);

                                        for ( i = 0; i < len ;i++)
                                         {///{6}
                                                OutPut[i]=0;
                                         }///{6}
                                         //itoa (T,output_file,10);
                                        sprintf(output_file, "%d", T);
                                        strcat(OutPut,Initial);
                                        strcat(OutPut,"_Velocity_Initial_");
                                        strcat(OutPut,output_file);
                                        strcat(OutPut,".txt");
                                        //printf("%s\n",OutPut);
                                        OutPut_File=OutPut;
```

```
afile=fopen(OutPut_File,"r");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{6}
            OutPut[i]=0;
 }//{6}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Angle_Initial_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
tfile=fopen(OutPut_File,"r");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{6}
            OutPut[i]=0;
 }//{6}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Anguler_Velocity_Initial_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
ofile=fopen(OutPut_File,"r");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{6}
            OutPut[i]=0;
 }//{6}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Electron_Initial_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
wfile=fopen(OutPut_File,"r");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{6}
            OutPut[i]=0;
 }//{6}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Hole_Initial_");
```

```
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
hfile=fopen(OutPut_File,"r");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{6}
        OutPut[i]=0;
 }//{6}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Time_Initial_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
tmfile=fopen(OutPut_File,"r");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{6}
        OutPut[i]=0;
 }//{6}


/*****************************************************************************/
rewind(yfile);
rewind(afile);
rewind(tfile);
rewind(ofile);

for (j=0;j<N;j++)
{//{6}

    if(j<200)
    {
        fscanf (yfile, "%f", &CH);
        BP_MAIN[j].Y=CH;
        fscanf (afile, "%f", &CH);
        BP_MAIN[j].A=CH;
        fscanf (tfile, "%f", &CH);
        BP_MAIN[j].TH=CH;
        fscanf (ofile, "%f", &CH);
        BP_MAIN[j].OM=CH;
        BP_MAIN[j].XR=0;
        BP_MAIN[j].XI=0;
        BP_MAIN[j].HR=0;
        BP_MAIN[j].HI=0;
    }
    else
    {  Position=256;
        while (abs(Position)>200)
        {
```

```
Position=fabs(Normal(0,1))*200;
                                                    }

BP_MAIN[j].Y=BP_MAIN[Position].Y;

BP_MAIN[j].A=BP_MAIN[Position].A;

BP_MAIN[j].TH=BP_MAIN[Position].TH;

BP_MAIN[j].OM=BP_MAIN[Position].OM;
                                                    BP_MAIN[j].XR=0;
                                                    BP_MAIN[j].XI=0;
                                                    BP_MAIN[j].HR=0;
                                                    BP_MAIN[j].HI=0;


                                                }


                                            }///{6}


                                                fclose(yfile);
                                                fclose(afile);
                                                fclose(tfile);
                                                fclose(ofile);
                                                fclose(wfile);
                                                fclose(hfile);
                                                fclose(tmfile);


                                        }///{5}
                                        else
                                        {///{5}
                                            if (T==Initial_Temperature)
                                            {///{6}
                                                for (j=0;j<N;j++)
                                                {///{7}
                                                    BP_MAIN[j].Y=0;
                                                    Parati=Normal(0, 1);
                                                    if (Parati>0)
                                                    {///{8}

BP_MAIN[j].A=Maxwell_Boltzman(0.5,T,0);

                                                    }///{8}
                                                    else
                                                    {///{8}
                                                        BP_MAIN[j].A=-
Maxwell_Boltzman(0.5,T,0);
                                                    }///{8}
                                                    Parati=Normal(0, 1);
                                                    if (Parati>0)
                                                    {///{8}
```

```
                                        BP_MAIN[j].OM=
Maxwell_Boltzman(0.5,T,1);
                                    }//{8}
                                    else
                                    {//{8}
                                        BP_MAIN[j].OM=-
Maxwell_Boltzman(0.5,T,1);
                                    }//{8}
                                    BP_MAIN[j].TH=0;
                                }//{7}

                            }//{6}
                            else
                            {//{6}
                                T=T-Temperature_Interval;
                                        if (DEBUG==0)
                                        {//{7}
                                            Initial=argv[1];
                                        }//{7}
                                        else
                                        {//{7}
                                            Initial="L.1";
                                        }//{7}

                                sprintf(output_file, "%d", T);
                                strcat(OutPut,Initial);
                                strcat(OutPut,"_Displacement_Initial_");
                                strcat(OutPut,output_file);
                                strcat(OutPut,".txt");
                                printf("%s\n",OutPut);
                                OutPut_File=OutPut;
                                yfile=fopen(OutPut_File,"r");
                                len=strlen(OutPut);
                                 for ( i = 0; i < len ;i++)
                                 {//{7}
                                        OutPut[i]=0;
                                 }//{7}

                                 //itoa (T,output_file,10);
                                sprintf(output_file, "%d", T);
                                strcat(OutPut,Initial);
                                strcat(OutPut,"_Velocity_Initial_");
                                strcat(OutPut,output_file);
                                strcat(OutPut,".txt");
                                //printf("%s\n",OutPut);
                                OutPut_File=OutPut;
                                afile=fopen(OutPut_File,"r");
                                len=strlen(OutPut);
                                 for ( i = 0; i < len ;i++)
                                 {//{7}
                                        OutPut[i]=0;
                                 }//{7}

                                 //itoa (T,output_file,10);
                                sprintf(output_file, "%d", T);
                                strcat(OutPut,Initial);
```

```
strcat(OutPut,"_Angle_Initial_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
tfile=fopen(OutPut_File,"r");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);

strcat(OutPut,"_Anguler_Velocity_Initial_");

strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
ofile=fopen(OutPut_File,"r");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Electron_Initial_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
wfile=fopen(OutPut_File,"r");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Hole_Initial_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
hfile=fopen(OutPut_File,"r");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
```

```
                                        }//{7}

                                        //itoa (T,output_file,10);
                                        sprintf(output_file, "%d", T);
                                        strcat(OutPut,Initial);
                                        strcat(OutPut,"_Time_Initial_");
                                        strcat(OutPut,output_file);
                                        strcat(OutPut,".txt");
                                        //printf("%s\n",OutPut);
                                        OutPut_File=OutPut;
                                        tmfile=fopen(OutPut_File,"r");
                                        len=strlen(OutPut);
                                        for ( i = 0; i < len ;i++)
                                        {//{7}
                                                OutPut[i]=0;
                                        }//{7}


/*************************************************************************************/
                                        rewind(yfile);
                                        rewind(afile);
                                        rewind(tfile);
                                        rewind(ofile);

                                        for (j=0;j<N;j++)
                                        {//{7}


                                                        fscanf (yfile, "%f", &CH);
                                                        BP_MAIN[j].Y=CH;
                                                        fscanf (afile, "%f", &CH);
                                                        BP_MAIN[j].A=CH;
                                                        fscanf (tfile, "%f", &CH);
                                                        BP_MAIN[j].TH=CH;
                                                        fscanf (ofile, "%f", &CH);
                                                        BP_MAIN[j].OM=CH;
                                                        BP_MAIN[j].XR=0;
                                                        BP_MAIN[j].XI=0;
                                                        BP_MAIN[j].HR=0;
                                                        BP_MAIN[j].HI=0;



                                        }//{7}


                                                fclose(yfile);
                                                fclose(afile);
                                                fclose(tfile);
                                                fclose(ofile);
                                                fclose(wfile);
                                                fclose(hfile);
                                                fclose(tmfile);
                                T=T+Temperature_Interval;
```

```
                                                        }//{6}

                                                }//{5}

                                        }//{4}
                }//{3}

        MPI_Comm_size ( MPI_COMM_WORLD, &size );
        MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
        M=N/size;
        MPI_Barrier(MPI_COMM_WORLD);
        MPI_Scatter
(&BP_MAIN[0],DIM*M,MPI_DOUBLE,&BP[ghost],DIM*M,MPI_DOUBLE,0,MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        if (Parallel_debug)
        {//{3}
                printf(" I have reached SCATT point %d \n",myrank);
        }//{3}


        Send_count_start=M;
        Destination=myrank+1;
        if (Destination>size-1)
        {//{3}
                Destination=0;
        }//{3}

        Origin=myrank-1;
        if (Origin<0)
        {//{3}
                Origin=size-1;
        }//{3}
```

```
        MPI_Sendrecv( &BP[Send_count_start], DIM*ghost,MPI_DOUBLE,Destination,4,&BP[0],
DIM*ghost,MPI_DOUBLE, Origin,4,MPI_COMM_WORLD, &M_status);
        MPI_Barrier(MPI_COMM_WORLD);



        Send_count_start=ghost;


        Destination=myrank-1;
        if (Destination<0)
        {//{3}
                Destination=size-1;
        }//{3}

        Origin=myrank+1;
        if (Origin>size-1)
        {//{3}
                Origin=0;
        }//{3}

  MPI_Sendrecv(&BP[Send_count_start],DIM*ghost,MPI_DOUBLE,Destination,3, &BP[M+ghost],
DIM*ghost,MPI_DOUBLE, Origin,3,MPI_COMM_WORLD, &M_status);
  MPI_Barrier(MPI_COMM_WORLD);
        if (Parallel_debug)
        {//{3}
                printf(" I have reached DIS point %d \n",myrank);
        }//{3}

                        if (myrank==0)
                        {//{3}
                                if (Restart_Switch)
                                {//{4}
                /*************************************************************************/
                /*
                Creat Input Files
                */
                /*************************************************************************/


                                        sprintf(output_file, "%d", T);
                                        strcat(OutPut,Initial);
                                        strcat(OutPut,"_Restart_");
                                        strcat(OutPut,output_file);
                                        strcat(OutPut,".txt");
                                        OutPut_File=OutPut;
                                        status = stat(OutPut_File, &buffer);

                                                if (status==0)
                                                {//{5}
                                                        IN=fopen(OutPut_File,"r");
                                                }//{5}
                                                len=strlen(OutPut);
                                                 for ( i = 0; i < len ;i++)
                                                 {//{5}
```

```
                                        OutPut[i]=0;
                                    }//{5}


    /*************************************************************************/
                                    if (status==0)
                                    {//{5}
                                        while (!feof(IN))
                                        {//{6}

                                            for (j=0;j<DIM*N+1;j++)
                                            {//{7}
                                                fscanf (IN, "%f", &CH);
                                                BP[jj].Y=CH;
                                            }//{7}
                                                fscanf (IN, "\n");

                                        }//{6}
                                    fclose(IN);
                                    }//{5}
                                }//{4}
                            }//{3}

/*************************************************************************/
/*
Initialize Variables
*/
/*************************************************************************/

        Kb=8.617343e-5;
        gma=.005;
        m=300*1.0358e-4;
        Energy_Tol=5e-8;

/*************************************************************************/
/*
Bring system into thermal equlibrium
*/
/*************************************************************************/
 m=1000;
h=1e-4;

E1=0;
E2=0;
Energy_Min=1;
/*************************************************************************/
/*
Allow Faster relazation
*/
/*************************************************************************/
gma=.5;

/*************************************************************************/
/*
Back to normal
*/
```

```
/*************************************************************************/
gma=0.005;


/*************************************************************************/
/*
Initialize Variables
*/
/*************************************************************************/

                n=1000000000;
        THO1=200;
        QW1=-0.4/0.04;
        //QD=2;
        V=0.1;
        //X=0.6;
//      h = ( tn - t0 ) / ( double ) ( n );
        h=1e-4;
        E_THO_CONST(E,THO,QD,QW1,THO1);


                        if (myrank==1)
                        {//{3}

                                printf ("Boltzman Constane   :");
                                printf ( "%20.15f\n", Kb );
                                printf ("Gamma               :");
                                printf ( "%20.15f\n", gma );
                                printf ("Chi                 :");
                                printf ( "%20.15f\n", X );
                                printf ("Transfer Integrals  :");
                                printf ( "%20.15f\n", V );
                                printf ("Well Depth          :");
                                printf ( "%20.15f\n", QW1 );
                                printf ("Well Separation     :");
                                printf ( "%d\n    ", QD );
                                printf ("Relax Time     :");
                                printf ( "%f\n    ", Relax_Time );
                                printf ("Rediation Count     :");
                                printf ( "%d\n    ", Red_Count );
                                printf ("Transport_Study     :");
                                printf ( "%d\n    ", Transport_Study );
                                printf ("Radiation_Study     :");
                                printf ( "%d\n    ", Radiation_Study );
                                printf ("Energy_Correction_Interval     :");
                                printf ( "%d\n    ", Energy_Correction_Interval );
                                printf ("Syncronization_Interval     :");
                                printf ( "%d\n    ", Syncronization_Interval );
                                printf ("Temperature     :");
                                printf ( "%d\n    ", T);
                                printf ("Tresh_hold     :");
                                printf ( "%f\n    ", Tresh_hold);
                                printf ("Time Step     :");
                                printf ( "%f\n    ", h);
                                printf ("Seed     :");
                                printf ( "%d\n    ", seed);
```

```
                printf ("Buffer_Size     :");
                printf ( "%d\n    ", Buffer_Size);
                printf ("Print_Interval    :");
                printf ( "%d\n    ", Print_Interval);
                printf ("Tresh_hold_Evoluation    :");
                printf ( "%f\n    ", Tresh_hold_Evoluation);
        }//{3}

/*************************************************************************/
/*
place a charge on to the system
 */
/*************************************************************************/



        if (myrank==0)
        {//{3}
                if (!Restart_Switch)
                {//{4}
                        if (DIM==4)
                        {//{5}
                                BP[N/2-1].XR=1;
                        }//{5}
                        else
                        {//{5}

                        }//{5}

                }//{4}
                else
                {//{4}
                        if (status==0)
                        {//{5}
                        //          t=y[DIM*N];
                        }//{5}
                }//{4}
                if (Restart_Switch)
                {//{4}

                                        if (DEBUG==0)
                                        {//{5}

                                                Initial=argv[1];
                                        }//{5}
                                        else
                                        {//{5}
                                                Initial="L.1";
                                        }//{5}

                                        //itoa (T,output_file,10);
                                        sprintf(output_file, "%d", T);
                                        strcat(OutPut,Initial);
                                        strcat(OutPut,"_OutPut_");
                                        strcat(OutPut,output_file);
```

```
                                        strcat(OutPut,".txt");
                                        //printf("%s\n",OutPut);
                                        OutPut_File=OutPut;
                                        if (status==0)
                                        {///{5}
                                                pfile=fopen(OutPut_File,"w+");
                                                i=0;
                                                Escaped=1;
                                        }///{5}
                                        else
                                        {///{5}
                                                Escaped=0;
                                                if(T==Final_Temperature)
                                                {///{6}
                                                        compleate=0;
                                                }///{6}
                                        }///{5}
                        }///{4}
                        else
                        {///{4}

/*****************************************************************************/
                /*
                Creat Output Files
                */

/*****************************************************************************/
                        if (Initial_Given)
                        {///{5}
                                        if (DEBUG==0)
                                                {///{6}
                                                Initial=argv[1];
                                                }///{6}
                                                else
                                                {///{6}
                                                Initial="L.1";
                                                }///{6}
                                if (myrank==0)
                                {///{6}

//Creat_file_structure(T,Initial,yfile,afile,tfile,ofile,wfile,hfile,tmfile);

                                for ( i = 0; i < len ;i++)
                                 {///{7}
                                        OutPut[i]=0;
                                 }///{7}
                                        //itoa (T,output_file,10);
                                sprintf(output_file, "%d", T);
                                strcat(OutPut,Initial);
                                strcat(OutPut,"_Displacement_");
                                strcat(OutPut,output_file);
                                strcat(OutPut,".txt");
                                //printf("%s\n",OutPut);
                                OutPut_File=OutPut;
                                yfile=fopen(OutPut_File,"w+");
                                len=strlen(OutPut);
```

```
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Velocity_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
afile=fopen(OutPut_File,"w+");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Angle_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
tfile=fopen(OutPut_File,"w+");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Anguler_Velocity_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
ofile=fopen(OutPut_File,"w+");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Electron_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
```

```
//printf("%s\n",OutPut);
OutPut_File=OutPut;
wfile=fopen(OutPut_File,"w+");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Hole_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
hfile=fopen(OutPut_File,"w+");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

  //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_TME_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
tmfile=fopen(OutPut_File,"w+");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Time_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
pfile=fopen(OutPut_File,"w+");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}


}//{6}
```

```
}//{5}
else
{//{5}
                        if (DEBUG==0)
                        {//{6}
                                Initial=argv[1];
                        }//{6}
                        else
                        {//{6}
                                Initial="L.1";
                        }//{6}
        if (myrank==0)
        {//{6}

//Creat_file_structure(T,Initial,yfile,afile,tfile,ofile,wfile,hfile,tmfile);


                                //itoa (T,output_file,10);
                        sprintf(output_file, "%d", T);
                        strcat(OutPut,Initial);
                        strcat(OutPut,"_Displacement_Initial_");
                        strcat(OutPut,output_file);
                        strcat(OutPut,".txt");
                        //printf("%s\n",OutPut);
                        OutPut_File=OutPut;
                        yfile=fopen(OutPut_File,"w+");
                        len=strlen(OutPut);
                         for ( i = 0; i < len ;i++)
                         {//{7}
                                 OutPut[i]=0;
                         }//{7}

                         //itoa (T,output_file,10);
                        sprintf(output_file, "%d", T);
                        strcat(OutPut,Initial);
                        strcat(OutPut,"_Velocity_Initial_");
                        strcat(OutPut,output_file);
                        strcat(OutPut,".txt");
                        //printf("%s\n",OutPut);
                        OutPut_File=OutPut;
                        afile=fopen(OutPut_File,"w+");
                        len=strlen(OutPut);
                         for ( i = 0; i < len ;i++)
                         {//{7}
                                 OutPut[i]=0;
                         }//{7}

                         //itoa (T,output_file,10);
                        sprintf(output_file, "%d", T);
                        strcat(OutPut,Initial);
                        strcat(OutPut,"_Angle_Initial_");
                        strcat(OutPut,output_file);
                        strcat(OutPut,".txt");
                        //printf("%s\n",OutPut);
                        OutPut_File=OutPut;
                        tfile=fopen(OutPut_File,"w+");
```

```
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Anguler_Velocity_Initial_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
ofile=fopen(OutPut_File,"w+");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Electron_Initial_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
wfile=fopen(OutPut_File,"w+");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Hole_Initial_");
strcat(OutPut,output_file);
strcat(OutPut,".txt");
//printf("%s\n",OutPut);
OutPut_File=OutPut;
hfile=fopen(OutPut_File,"w+");
len=strlen(OutPut);
 for ( i = 0; i < len ;i++)
 {//{7}
        OutPut[i]=0;
 }//{7}

 //itoa (T,output_file,10);
sprintf(output_file, "%d", T);
strcat(OutPut,Initial);
strcat(OutPut,"_Time_Initial_");
strcat(OutPut,output_file);
```

```
                                        strcat(OutPut,".txt");
                                        //printf("%s\n",OutPut);
                                        OutPut_File=OutPut;
                                        tmfile=fopen(OutPut_File,"w+");
                                        len=strlen(OutPut);
                                         for ( i = 0; i < len ;i++)
                                         {//{7}
                                                OutPut[i]=0;
                                         }//{7}
                                    }//{6}
                                }//{5}

                                }//{4}
                            }//{3}

                    if(Radiation_Study)
                                    {//{3}
                                                if (myrank==0)
                                                {//{4}
                                                Radiation_Location=N*fabs(Normal(0,1));
                                                if (Radiation_Location<N)
                                                {//{5}
                                                        nh = div(Radiation_Location, M);
                                                        Ray.Sub_unit=nh.quot;
                                                        Ray.Position=nh.rem;
                                                        Ray.Ray_Hit=1;
                                                }//{5}
                                                }//{4}

                                        MPI_Bcast(&Ray,3,MPI_INT,0,MPI_COMM_WORLD);
                                        if (Ray.Ray_Hit==1)
                                        {//{4}
                                                Ray.Ray_Hit=0;
                                                if (myrank==Ray.Sub_unit)
                                                {//{5}
                                                BP[Ray.Position].XR=1;
                                                BP[Ray.Position].HR=1;
                                                }//{5}
                                        }//{4}
                                    }//{3}
Radiation_Hit=1;
t=0;

i=0;
count=0;
Escaped=1;
Norm1=0;

if (Parallel_debug)
{//{3}
        printf(" I have reached INIT point %d \n",myrank);
}//{3}


        for (j=0;j<M+2*ghost;j++)
        {
```

```
            y0[j].HR=BP[j].HR;
            y0[j].XR=BP[j].XR;
            y0[j].HI=BP[j].HI;
            y0[j].XI=BP[j].XI;
            y0[j].Y=BP[j].Y;
            y0[j].A=BP[j].A;
            y0[j].OM=BP[j].OM;
            y0[j].TH=BP[j].TH;

        }

        for (j=0;j<M+2*ghost;j++)
        {
            ytemp[j].HR=BP[j].HR;
            ytemp[j].XR=BP[j].XR;
            ytemp[j].HI=BP[j].HI;
            ytemp[j].XI=BP[j].XI;
            ytemp[j].Y=BP[j].Y;
            ytemp[j].A=BP[j].A;
            ytemp[j].OM=BP[j].OM;
            ytemp[j].TH=BP[j].TH;

        }

Stochastic_RK4 ( ytemp, Strand,X,V,t, h, q, &seed ,T,gma,Total_Energy,Transport_Study,Job_ID);


        for (j=0;j<M+2*ghost;j++)
        {
            y1[j].HR=ytemp[j].HR;
            y1[j].XR=ytemp[j].XR;
            y1[j].HI=ytemp[j].HI;
            y1[j].XI=ytemp[j].XI;
            y1[j].Y=ytemp[j].Y;
            y1[j].A=ytemp[j].A;
            y1[j].OM=ytemp[j].OM;
            y1[j].TH=ytemp[j].TH;

        }

         Send_count_start=M;
         Destination=myrank+1;
         if (Destination>size-1)
         {//{3}
                Destination=0;
         }//{3}

         Origin=myrank-1;
         if (Origin<0)
         {//{3}
                Origin=size-1;
         }//{3}
```

```
        MPI_Sendrecv( &ytemp[Send_count_start], DIM*ghost,MPI_DOUBLE,Destination,4,&ytemp[0],
DIM*ghost,MPI_DOUBLE, Origin,4,MPI_COMM_WORLD, &M_status);
        MPI_Barrier(MPI_COMM_WORLD);



        Send_count_start=ghost;


        Destination=myrank-1;
        if (Destination<0)
        {//{3}
                Destination=size-1;
        }//{3}

        Origin=myrank+1;
        if (Origin>size-1)
        {//{3}
                 Origin=0;
        }//{3}

   MPI_Sendrecv(&ytemp[Send_count_start],DIM*ghost,MPI_DOUBLE,Destination,3, &ytemp[M+ghost],
DIM*ghost,MPI_DOUBLE, Origin,3,MPI_COMM_WORLD, &M_status);
   MPI_Barrier(MPI_COMM_WORLD);



        Stochastic_RK4 ( ytemp, Strand,X,V,t, h, q, &seed ,T,gma,Total_Energy,Transport_Study,Job_ID);
        for (j=0;j<M+2*ghost;j++)
        {
                y2[j].HR=ytemp[j].HR;
                y2[j].XR=ytemp[j].XR;
                y2[j].HI=ytemp[j].HI;
                y2[j].XI=ytemp[j].XI;
                y2[j].Y=ytemp[j].Y;
                y2[j].A=ytemp[j].A;
                y2[j].OM=ytemp[j].OM;
                y2[j].TH=ytemp[j].TH;

        }


        Send_count_start=M;
        Destination=myrank+1;
        if (Destination>size-1)
        {//{3}
                Destination=0;
        }//{3}

        Origin=myrank-1;
        if (Origin<0)
        {//{3}
                 Origin=size-1;
        }//{3}
```

```
        MPI_Sendrecv( &ytemp[Send_count_start], DIM*ghost,MPI_DOUBLE,Destination,4,&ytemp[0],
DIM*ghost,MPI_DOUBLE, Origin,4,MPI_COMM_WORLD, &M_status);
        MPI_Barrier(MPI_COMM_WORLD);



        Send_count_start=ghost;
        Destination=myrank-1;
        if (Destination<0)
        {//{3}
                Destination=size-1;
        }//{3}

        Origin=myrank+1;
        if (Origin>size-1)
        {//{3}
                Origin=0;
        }//{3}

  MPI_Sendrecv(&ytemp[Send_count_start],DIM*ghost,MPI_DOUBLE,Destination,3, &ytemp[M+ghost],
DIM*ghost,MPI_DOUBLE, Origin,3,MPI_COMM_WORLD, &M_status);
  MPI_Barrier(MPI_COMM_WORLD);
        Stochastic_RK4 ( ytemp, Strand,X,V,t, h, q, &seed ,T,gma,Total_Energy,Transport_Study,Job_ID);
        for (j=0;j<M+2*ghost;j++)
        {
                BP[j].HR=ytemp[j].HR;
                BP[j].XR=ytemp[j].XR;
                BP[j].HI=ytemp[j].HI;
                BP[j].XI=ytemp[j].XI;
                BP[j].Y=ytemp[j].Y;
                BP[j].A=ytemp[j].A;
                BP[j].OM=ytemp[j].OM;
                BP[j].TH=ytemp[j].TH;

        }

        Send_count_start=M;
        Destination=myrank+1;
        if (Destination>size-1)
        {//{3}
                Destination=0;
        }//{3}

        Origin=myrank-1;
        if (Origin<0)
        {//{3}
                Origin=size-1;
        }//{3}

        MPI_Sendrecv( &BP[Send_count_start], DIM*ghost,MPI_DOUBLE,Destination,4,&BP[0],
DIM*ghost,MPI_DOUBLE, Origin,4,MPI_COMM_WORLD, &M_status);
        MPI_Barrier(MPI_COMM_WORLD);
        Send_count_start=ghost;
        Destination=myrank-1;
```

```
        if (Destination<0)
        {///{3}
                Destination=size-1;
        }///{3}

        Origin=myrank+1;
        if (Origin>size-1)
        {///{3}
                Origin=0;
        }///{3}

   MPI_Sendrecv(&BP[Send_count_start],DIM*ghost,MPI_DOUBLE,Destination,3, &BP[M+ghost],
DIM*ghost,MPI_DOUBLE, Origin,3,MPI_COMM_WORLD, &M_status);
   MPI_Barrier(MPI_COMM_WORLD);
 Buffer_Print_Advance=1;
while (Escaped==1) //for ( i = 1; i <= n; i++ )
{///{3}

        t=t+h;
        i=i+1;

        count=count+h;

        Energy_Correction_Counter=Energy_Correction_Counter+1;
        if(Transport_Study)
        {///{4}
                if (Energy_Correction_Interval<Energy_Correction_Counter)
                {///{5}
                                Energy_Correction_Counter=0;
                                Energy_Calculation(BP,Total_Energy,X,Transport_Study,Job_ID);

        Onsite_Energy_Calculation(BP,Total_Energy,X,Hydration_Energy,Transport_Study);
                }///{5}
        }///{4}


        //Runge_Kutta_Fehlberg (BP,y2,y1,y0,h, t,Strand,X,V,T,gma,Total_Energy,Transport_Study,Job_ID);
        Stochastic_RK4(BP,Strand,X,V,t,h,q,&seed,T,gma,Total_Energy,Transport_Study,Job_ID);
        nh = div(i, 1000);
        Buffer_Print_Interval=nh.rem;
        nh = div(Buffer_Print_Advance, Buffer_Size);
        Buffer_Index=nh.rem;
        if (Buffer_Print_Interval==1)
        {
                Buffer_Print_Advance=Buffer_Print_Advance+1;
                for (j=ghost;j<M+ghost;j++)
                {
                        Local_Index=Buffer_Index*M+(j-ghost);

                        Output_Buffer[Local_Index].Wave_Function=pow(BP[j].XI,2)+pow(BP[j].XR,2);
                        Output_Buffer[Local_Index].Displacement=BP[j].Y;
                        Output_Buffer[Local_Index].Twist_Angle=BP[j].TH;

                }
        }
```

```
                        if (i>n)
                        {///{4}

/*************************************************************************/
                        /*
                        Creat Restart Files
                        */

/*************************************************************************/
                                if (myrank==0)
                                {///{5}
                                        if (DEBUG==0)
                                        {///{6}
                                        Initial=argv[1];
                                        }///{6}
                                        else
                                        {///{6}
                                                Initial="L.1";
                                        }///{6}


                                        //itoa (T,output_file,10);
                                        sprintf(output_file, "%d", T);
                                        strcat(Prefix,Initial);
                                        strcat(Prefix,"_Restart_");
                                        strcat(Prefix,output_file);
                                        strcat(Prefix,".txt");
                                        //printf("%s\n",Prefix);
                                        Restart_File=Prefix;
                                        Restart=fopen(Restart_File,"w+");
                                }///{5}

/*************************************************************************/
                                Escaped=0;

/*************************************************************************/
                        /*
                        set restart points
                        */

/*************************************************************************/
                                if (myrank==0)
                                {///{5}
                                        for (j=0;j<DIM*N+1;j++)
                                        {///{6}

                                                        if (j<DIM*N)
                                                        {///{7}
                                                //      fprintf(Restart,"%20.16f
",y[j]);

                                                        }///{7}
                                                        else
                                                        {///{7}
                                                //
        fprintf(Restart,"%20.16f  ",t);
```

```
                                                                        }//{7}


                                                                }//{6}
                                                                fprintf(Restart,"\n");
                                                                fclose(Restart);
                                                                len=strlen(Prefix);
                                                                printf("%d\n",len);
                                                                 for ( j = 0; j < len ;j++)
                                                                 {//{6}
                                                                        Prefix[j]=0;
                                                                 }//{6}
                                                        }//{5}
                                                }//{4}
/*************************************************************************/
/*
Perform the computations
*/
/*************************************************************************/
                                                if(Radiation_Study)
                                                {//{4}
                                                if (Relax_Time<t)
                                                        {//{5}
                                                                if (count>radiation_rate)
                                                                {//{6}
                                                                        count=0;
                                                                        for (j=0;j<Red_Count;j++)
                                                                        {//{7}

                                                                                        if (myrank==0)
                                                                                        {//{8}

        Radiation_Location=N*fabs(Normal(0,1));
                                                                                        if
(Radiation_Location<N)
                                                                                        {//{9}
                                                                                                nh =
div(Radiation_Location, M);

        Ray.Sub_unit=nh.quot;

        Ray.Position=nh.rem;

        Ray.Ray_Hit=1;
                                                                                        }//{9}
                                                                                        }//{8}


        MPI_Bcast(&Ray,3,MPI_INT,0,MPI_COMM_WORLD);
                                                                                if (Ray.Ray_Hit==1)
                                                                                {//{8}
                                                                                        Ray.Ray_Hit=0;
                                                                                        if
(myrank==Ray.Sub_unit)
                                                                                        {//{9}
                                                                                        BP[Ray.Position].XR=1;
```

```
                                                                BP[Ray.Position].HR=1;
                                                                }//{9}
                                                        }//{8}

                                                        }//{7}
                                                }//{6}
                                        }//{5}
                                }//{4}


                                if (Relax_Time<t)
                                {//{4}
                                        if (Radiation_Hit)
                                        {//{5}
                                                Radiation_Hit=0;
                                                if(Transport_Study)
                                                {//{6}

                                                        if (myrank==0)
                                                        {//{7}
                                                        Radiation_Location=N/2-1;
                                                                if (Radiation_Location<N)
                                                                {//{8}
                                                                        nh =
div(Radiation_Location, M);
                                                                        Ray.Sub_unit=nh.quot;
                                                                        Ray.Position=nh.rem;
                                                                        Ray.Ray_Hit=1;
                                                                        nh =
div(Radiation_Location, M);
                                                                        cell_donor.Initiate=1;

        cell_donor.Sub_unit=nh.quot;
        cell_donor.Position=nh.rem;
        base_donor.well_depth=QW1;
        base_donor.absorption=0;
        nh = div(Radiation_Location-QD, M);
        cell_acceptor.Initiate=1;
        cell_acceptor.Sub_unit=nh.quot;
        cell_acceptor.Position=nh.rem;
        base_acceptor.well_depth=QW1;
        base_acceptor.absorption=1000;
        }//{8}
        }//{7}

        MPI_Bcast(&Ray,3,MPI_INT,0,MPI_COMM_WORLD);
        MPI_Bcast(&cell_acceptor,3,MPI_INT,0,MPI_COMM_WORLD);
        MPI_Bcast(&cell_donor,3,MPI_INT,0,MPI_COMM_WORLD);
        MPI_Bcast(&base_acceptor,2,MPI_DOUBLE,0,MPI_COMM_WORLD);
        MPI_Bcast(&base_donor,2,MPI_DOUBLE,0,MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
                if (Ray.Ray_Hit==1)
                {//{7}
                Ray.Ray_Hit=0;
                if (myrank==Ray.Sub_unit)
                {//{8}
```

```
                BP[Ray.Position+ghost-1].XR=1;
                }//{8}
                }//{7}
                if (cell_donor.Initiate==1)
                {//{7}
                if (myrank==cell_donor.Sub_unit)
                {//{8}

        Strand[cell_donor.Position+ghost-1].well_depth=base_donor.well_depth;
        Strand[cell_donor.Position+ghost-1].absorption=base_donor.absorption;
        //printf(" base_donor.well_depth %f \n cell_donor.Sub_unit %d \n base_donor.absorption %f \n
cell_donor.Position+ghost %d \n myrank
%d",base_donor.well_depth,cell_donor.Sub_unit,base_donor.absorption,cell_donor.Position+ghost-1,myrank);
        }//{8}
        }//{7}
        if (cell_acceptor.Initiate==1)
        {//{7}
        if (myrank==cell_acceptor.Sub_unit)
        {//{8}
        Strand[cell_acceptor.Position+ghost-1].well_depth=base_acceptor.well_depth;
        Strand[cell_acceptor.Position+ghost-1].absorption=base_acceptor.absorption;
        //printf(" base_acceptor.well_depth %f \n cell_acceptor.Sub_unit %d \n base_acceptor.absorption
%f \n cell_acceptor.Position+ghost %d \n myrank
%d",base_acceptor.well_depth,cell_acceptor.Sub_unit,base_acceptor.absorption,cell_acceptor.Position+ghost-
1,myrank);
        }//{8}
        }//{7}
}//{6}
}//{5}
}//{4}




/*****************************************************************************/
//Call Computation
/*****************************************************************************/

nh = div(i, Print_Interval);
if (nh.rem==(1))
{//{4}
MPI_Gather ( &BP[ghost], DIM*M, MPI_DOUBLE, &BP_MAIN[0], DIM*M, MPI_DOUBLE,0,
MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
if (Parallel_debug)
{//{5}
printf(" I have reached GATHE point %d \n",myrank);
}//{5}
}//{4}
if (Printing_On)
{
nh = div(i, Print_Interval);
if (nh.rem==(1))
{//{4}
if (Initial_Given)
```

```
{///{5}
                                        if (myrank==0)
                                        {///{6}
                                        fprintf(tmfile,"%f \n",t);
                                        fflush(tmfile) ;
                                        Print_matrix(BP_MAIN,yfile,afile,tfile,ofile,wfile,hfile);
                                        }///{6}
                                        Norm1=0;
                                        Exit_Switch=0;
                                        EXIT_ON=0;
                                        Exit_Conditions=Norm_Calculation(BP,cell_donor);

                                        //if (myrank==cell_donor.Sub_unit)
                                        //{///{6}

                                                        Exit_Switch=0;
                                                        if (Exit_Conditions.Return_Probability<Tresh_hold)
                                                        {


        if((1/Exit_Conditions.Participation_Function)>Tresh_hold_Evoluation)
        {
        //if(Exit_Conditions.Shannon_Information_Entrophy>1)
        //{

        if (myrank==0)
        {
        if (Relax_Time<t)
        {///{6}
        Exit_Switch=1;
        }
        }
        //}
        }
        }
        MPI_Bcast(&Exit_Switch,1,MPI_INT,0,MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        if (Exit_Switch)
        {///{7}
        //Pre_Escape1=0;
        //if ((Pre_Escape1==0)&&(Pre_Escape2==0))
        //{///{8}

        MPI_Gather ( &Output_Buffer[0], Buffer_Size*3*M, MPI_DOUBLE, &Master_Output_Buffer[0],
Buffer_Size*3*M, MPI_DOUBLE,0, MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        if (myrank==0)
         {///{9}
        printf("%d  %d\n",Buffer_Size,Buffer_Index);

Final_Times_Print_matrix(Master_Output_Buffer,yfile,tfile,wfile,Buffer_Size,Buffer_Index);
         Escaped=0;
                    }///{9}
        MPI_Bcast(&Escaped,1,MPI_INT,0,MPI_COMM_WORLD);
        MPI_Barrier(MPI_COMM_WORLD);
        //}///{8}
```

```
//Pre_Escape2=Pre_Escape1;
//}///{7}
//else
//{///{7}
//Pre_Escape2=1;
}///{7}
MPI_Bcast(&Escaped,1,MPI_INT,0,MPI_COMM_WORLD);
MPI_Barrier(MPI_COMM_WORLD);
/*for (j=cell_donor.Position+ghost-(abs(QD)+1);j<cell_donor.Position+ghost+(abs(QD)+2);j++)
                                        {///{7}
                                                Norm1=Norm1+(BP[j].XR*BP[j].XR+BP[j].XI*BP[j].XI);
                                        }///{7}
                                        if (Norm1<Tresh_hold)
                                        {///{7}
                                        if (Relax_Time<t)
                                                {///{8}
                                                        Exit_Switch=1;
                                                }///{8}
                                        }///{7}*/

                                //}///{6}
                        }///{5}
                        else
                        {///{5}
                                if (Relax_Time<t)
                                {///{6}
                                        if (myrank==0)
                                        {///{7}
                                                fprintf(tmfile,"%f \n",t);
                                                fflush(tmfile) ;
                                                Print_matrix(BP_MAIN,yfile,afile,tfile,ofile,wfile,hfile);
                                        }///{7}
                                         Escaped=0;
                                         MPI_Bcast(&Escaped,1,MPI_INT,0,MPI_COMM_WORLD);
                                        MPI_Barrier(MPI_COMM_WORLD);
                                }///{6}
                        }///{5}
                }///{4}
                else
                {///{4}
                        if (Initial_Given)
                        {///{5}
                                Norm1=0;
                                Exit_Switch=0;
                                /*      if (myrank==cell_donor.Sub_unit)
                                        {///{6}
                                        for (j=cell_donor.Position+ghost-
(abs(QD)+1);j<cell_donor.Position+ghost+(abs(QD)+2);j++)
                                        {///{7}
                                                Norm1=Norm1+(BP[j].XR*BP[j].XR+BP[j].XI*BP[j].XI);
                                        }///{7}
                                        if (Norm1<Tresh_hold)
                                        {///{7}
                                        Exit_Switch=1;
                                        }///{7}
```

```
}///{6}*/
Exit_Conditions=Norm_Calculation(BP,cell_donor);

        if (myrank==0)
        {

                if (Exit_Conditions.Return_Probability<Tresh_hold)
                {


if((1/Exit_Conditions.Participation_Function)>Tresh_hold_Evoluation)
                                {

//if(Exit_Conditions.Shannon_Information_Entrophy>1)
                                        //{
                                                if (Relax_Time<t)
                                                {///{6}
                                                Exit_Switch=1;

                                                }

                                        //}
                                }
                        }
                }



        //nh = div(i, Syncronization_Interval);
        //if (nh.rem==(1))
        //{///{6}
                MPI_Bcast(&Exit_Switch,1,MPI_INT,0,MPI_COMM_WORLD);
                MPI_Barrier(MPI_COMM_WORLD);
                if (Parallel_debug)
                {///{3}
                        if (myrank==0)
                        {
                        printf("%f %f %f %d %d %f %f%f %f\n",Exit_Conditions.Return_Probability,1/Exit_Conditions.Participation_Function,Exit_Conditions.Shannon_Information_Entrophy,Escaped,Exit_Switch,Tresh_hold,Tresh_hold_Evoluation,Relax_Time,t);
                        }
                }
                if (Exit_Switch)
                {///{7}


                        //Pre_Escape1=0;
                        //if ((Pre_Escape1==0)&&(Pre_Escape2==0))
                        //{///{8}

                                MPI_Gather ( &BP[ghost], DIM*M, MPI_DOUBLE, &BP_MAIN[0], DIM*M, MPI_DOUBLE,0, MPI_COMM_WORLD);
                                MPI_Barrier(MPI_COMM_WORLD);
                                MPI_Gather ( &Output_Buffer[0], Buffer_Size*3*M, MPI_DOUBLE, &Master_Output_Buffer[0], Buffer_Size*3*M, MPI_DOUBLE,0, MPI_COMM_WORLD);
```

```
                                                MPI_Barrier(MPI_COMM_WORLD);
                                                if (myrank==0)
                                                 {//{9}
                                                        Escaped=0;
                                                        fprintf(tmfile,"%f \n",t);
                                                        fflush(tmfile) ;

                                                                printf("%d
%d\n",Buffer_Size,Buffer_Index);

Final_Times_Print_matrix(Master_Output_Buffer,yfile,tfile,wfile,Buffer_Size,Buffer_Index);

Print_matrix(BP_MAIN,yfile,afile,tfile,ofile,wfile,hfile);



                                                }//{9}

        MPI_Bcast(&Escaped,1,MPI_INT,0,MPI_COMM_WORLD);
                                                MPI_Barrier(MPI_COMM_WORLD);

                                        //}//{8}
                                        //Pre_Escape2=Pre_Escape1;
                                        Exit_Switch=0;

                                //}//{7}
                                //else
                                //{//{7}
                                //Pre_Escape2=1;

                                }//{7}
                        //}//{6}

                }//{5}
                else
                {//{5}
                        if (Relax_Time<t)
                        {//{6}
                                if (myrank==0)
                                {//{7}
                                        fprintf(tmfile,"%f \n",t);
                                        fflush(tmfile) ;
                                        Print_matrix(BP_MAIN,yfile,afile,tfile,ofile,wfile,hfile);
                                }//{7}
                                 Escaped=0;
                                 MPI_Bcast(&Escaped,1,MPI_INT,0,MPI_COMM_WORLD);
                                MPI_Barrier(MPI_COMM_WORLD);
                        }//{6}

                }//{5}
        }//{4}
        }


        if(T==Final_Temperature)
        {//{4}
```

```
                        compleate=0;
                }///{4}

//Escaped=1;

/*****************************************************************************/

 MPI_Barrier(MPI_COMM_WORLD);

/*****************************************************************************/
}///{3}//while Escaped
T=T+Temperature_Interval;

Elaped_time +=MPI_Wtime();
        if (myrank==1)
        {///{3}
                printf ("Elaped_time     :");
                printf ( "%f\n    ", Elaped_time);
        }///{3}



/*****************************************************************************/

                len=strlen(OutPut);
                for ( i = 0; i < len ;i++)
                {///{3}
                        OutPut[i]=0;
                }///{3}
                if (myrank==0)
                {///{3}

                        if (!Restart_Switch)
                        {///{4}

                                if (Initial_Given)
                                {///{5}

                                fclose(pfile);
                                fclose(yfile);
                                fclose(afile);
                                fclose(tfile);
                                fclose(ofile);
                                fclose(wfile);
                                fclose(hfile);
                                fclose(tmfile);
                                }
                                else
                                {
                                fclose(yfile);
                                fclose(afile);
                                fclose(tfile);
                                fclose(ofile);
                                fclose(wfile);
                                fclose(hfile);
                                fclose(tmfile);
```

```
                                        }


                                }///{4}
                                else
                                {///{4}
                                        if (status==0)
                                        {///{5}
                                        if (Initial_Given)
                                        {///{5}
                                                fclose(pfile);
                                        fclose(yfile);
                                        fclose(afile);
                                        fclose(tfile);
                                        fclose(ofile);
                                        fclose(wfile);
                                        fclose(hfile);
                                        fclose(tmfile);
                                        }
                                        else
                                        {

                                        fclose(yfile);
                                        fclose(afile);
                                        fclose(tfile);
                                        fclose(ofile);
                                        fclose(wfile);
                                        fclose(hfile);
                                        fclose(tmfile);
                                        }
                                        }///{5}
                                }///{4}
                        }///{3}

/*****************************************************************************/
}///{2}     //while compleate
/*****************************************************************************/

 MPI_Finalize();

/*****************************************************************************/
/*
Clear memory allocated for local variables
*/
/*****************************************************************************/
free(BP);
BP=NULL;
free(E);
E=NULL;
free(THO);
THO=NULL;
/*****************************************************************************/
  return 0;
}///{1}
/*****************************************************************************/
```

(ii) Compute Maxwell Boltzman Distribution

```
/******************************************************************************/
/*
  Purpose:
    Compute Maxwell Boltzman Distribution
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
   16 Aug 2010
  Parallel Version: C MPI
  Author:
          Neranjan Edirisinghe
/******************************************************************************/


#include "stochastic_rk.h"
double Maxwell_Boltzman(double G,int T,int S)
{
double value,X1,X2,X3,m,Kb,R0;
 m=300*1.0358e-4;
  Kb=8.617343e-5;
  R0=4.45*10;
  if (S==1)
  {
          m=m*R0;
  }
X1=Normal(0, sqrt(2*Kb*T*m*G));
X2=Normal(0, sqrt(2*Kb*T*m*G));
X3=Normal(0, sqrt(2*Kb*T*m*G));
value=sqrt(pow(X1,2)+pow(X2,2)+pow(X3,2));
return value;
}
```

(iii) Distribute data across the processors

```
/******************************************************************************/
/*
  Purpose:
    Distribute data across the processors
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
   16 May 2009
  Parallel Version: C MPI
  Author:
          Neranjan Edirisinghe
/******************************************************************************/


# include "stochastic_rk.h"
```

```
void Distribute_data(Base_Pair yk[],int Tag1,int Tag2,int Job_ID)
{
int myrank,size,M,Destination,Origin,Send_count_start;
MPI_Status status;

MPI_Comm_size ( MPI_COMM_WORLD, &size );
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
M=N/size;
Send_count_start=M;
                Destination=myrank+1;
        if (Destination>size-1)
        {
                Destination=0;
        }

        Origin=myrank-1;
        if (Origin<0)
        {
                Origin=size-1;
        }

   MPI_Sendrecv(&yk[Send_count_start],DIM*ghost,MPI_DOUBLE,Destination,Tag1,&yk[0],
DIM*ghost,MPI_DOUBLE, Origin,Tag1,MPI_COMM_WORLD, &status);
   MPI_Barrier(MPI_COMM_WORLD);
        Send_count_start=ghost;
        if (Send_count_start>M)
        {
                Send_count_start=0;
        }


        Destination=myrank-1;
        if (Destination<0)
        {
                Destination=size-1;
        }

        Origin=myrank+1;
        if (Origin>size-1)
        {
                Origin=0;
        }
        MPI_Sendrecv(&yk[Send_count_start],DIM*ghost,MPI_DOUBLE,Destination,Tag2,&yk[M+ghost],
DIM*ghost,MPI_DOUBLE, Origin,Tag2,MPI_COMM_WORLD, &status);

MPI_Barrier(MPI_COMM_WORLD);
}
```

(iv) Stachastic Differntail Equation solver Using fourth order Runge-Kutta method

```
/*****************************************************************************/
/*
  Purpose:
    Solve Stachastic Differntail Equation Using fourth order Runge-Kutta method
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
    07 July 2010
  Serial Version Author:
    John Burkardt
  Parallel Versiton Author:
          Neranjan Edirisinghe
*/
/*****************************************************************************/

#include "stochastic_rk.h"
double gv (int x,int T,double );
void Stochastic_RK4(Base_Pair BP[],Base Strand[],double X,double V,double t,double h,double q,int *seed,int
T,double G,double Total_Energy[],int Transport_Study,int Job_ID)
{
          double a21;
          double a31;
          double a32;
          double a41;
          double a42;
          double a43;
          double a51;
          double a52;
          double a53;
          double a54;

          double q1;
          double q2;
          double q3;
          double q4;
          double t1;
          double t2;
          double t3;
          double t4;
          double w1;
          double w2;
          double w3;
          double w4;


  int myrank,size,M,Send_count_start;

                    double *OLD;
          Base_Pair *dy,*yk,*K1,*K2,*K3,*K4;
                    int j;
                    double Kb,m;
  m=300*1.0358e-4;
```

```
 Kb=8.617343e-5;

/*****************************************************************************/
/**/

        MPI_Comm_size ( MPI_COMM_WORLD, &size );
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
M=N/size;
Send_count_start=M;
 //         printf(" I have reached INSIDE SK4 point %d \n",myrank);
/*****************************************************************************/
/*
Allocate memory for dy
*/
/*****************************************************************************/
                dy=malloc ((M+2*ghost)*sizeof(Base_Pair));
                if (dy == NULL)
                {
                        printf("error defining dy  ");
                }

                yk=malloc ((M+2*ghost)*sizeof(Base_Pair));
                if (dy == NULL)
                {
                        printf("error defining dy  ");
                }

                K1=malloc ((M+2*ghost)*sizeof(Base_Pair));
                if (dy == NULL)
                {
                        printf("error defining dy  ");
                }

                K2=malloc ((M+2*ghost)*sizeof(Base_Pair));
                if (dy == NULL)
                {
                        printf("error defining dy  ");
                }

                K3=malloc ((M+2*ghost)*sizeof(Base_Pair));
                if (dy == NULL)
                {
                        printf("error defining dy  ");
                }

                K4=malloc ((M+2*ghost)*sizeof(Base_Pair));
                if (dy == NULL)
                {
                        printf("error defining dy  ");
                }

                OLD=malloc ((M+2*ghost)*sizeof(double));
                if (OLD == NULL)
                {
                        printf("error defining OLD  ");
                }
```

```
/*****************************************************************************/

/*****************************************************************************/
/*
Variable Initialization
*/
/*****************************************************************************/

//printf("%f %d   \n",BP[1].Y,myrank);

                        for (j=0;j<(M+2*ghost);j++)
                        {
                                dy[j].HR=0.0;
                                dy[j].XR=0.0;
                                dy[j].HI=0.0;
                                dy[j].XI=0.0;
                                dy[j].Y=0.0;
                                dy[j].A=0.0;
                                dy[j].OM=0.0;
                                dy[j].TH=0.0;
                                K1[j].HR=0.0;
                                K1[j].XR=0.0;
                                K1[j].HI=0.0;
                                K1[j].XI=0.0;
                                K1[j].Y=0.0;
                                K1[j].A=0.0;
                                K1[j].OM=0.0;
                                K1[j].TH=0.0;
                                K2[j].HR=0.0;
                                K2[j].XR=0.0;
                                K2[j].HI=0.0;
                                K2[j].XI=0.0;
                                K2[j].Y=0.0;
                                K2[j].A=0.0;
                                K2[j].OM=0.0;
                                K2[j].TH=0.0;
                                K3[j].HR=0.0;
                                K3[j].XR=0.0;
                                K3[j].HI=0.0;
                                K3[j].XI=0.0;
                                K3[j].Y=0.0;
                                K3[j].A=0.0;
                                K3[j].OM=0.0;
                                K3[j].TH=0.0;
                                K4[j].HR=0.0;
                                K4[j].XR=0.0;
                                K4[j].HI=0.0;
                                K4[j].XI=0.0;
                                K4[j].Y=0.0;
                                K4[j].A=0.0;
                                K4[j].OM=0.0;
                                K4[j].TH=0.0;
                                yk[j].HR=0.0;
                                yk[j].XR=0.0;
                                yk[j].HI=0.0;
                                yk[j].XI=0.0;
```

```
                                yk[j].Y=0.0;
                                yk[j].A=0.0;
                                yk[j].OM=0.0;
                                yk[j].TH=0.0;


                        }

          a21 =   0.66667754298442;
          a31 =   0.63493935027993;
          a32 =   0.00342761715422;
          a41 = - 2.32428921184321;
          a42 =   2.69723745129487;
          a43 =   0.29093673271592;
          a51 =   0.25001351164789;
          a52 =   0.67428574806272;
          a53 = - 0.00831795169360;
          a54 =   0.08401868181222;

          q1 = 3.99956364361748;
          q2 = 1.64524970733585;
          q3 = 1.59330355118722;
          q4 = 0.26330006501868;
/*****************************************************************************/

/*****************************************************************************/

        for (j=ghost;j<M+ghost;j++)
        {
                yk[j].HR=BP[j].HR;
                yk[j].XR=BP[j].XR;
                yk[j].HI=BP[j].HI;
                yk[j].XI=BP[j].XI;
                yk[j].Y=BP[j].Y;
                yk[j].A=BP[j].A;
                OLD[j]=BP[j].OM;
                yk[j].OM=BP[j].OM;
                yk[j].TH=BP[j].TH;

        }
          Distribute_data(yk,21,22,Job_ID);
          RHS_FUN(yk,dy,Strand,X,V,G,OLD,h,Total_Energy,Transport_Study);
           //Distribute_data(yk,1,2,Job_ID);
        //printf(" I have reached TRANS point %d \n",myrank);
          t1=t;
          for (j=ghost;j<M+ghost;j++)
                {

                        w1 = Normal(0, sqrt(2*Kb*T*m*G)) * sqrt ( q1 * q / h );
                        K1[j].Y=h*dy[j].Y+h*w1;
                        //w1 = r8_normal_01 (seed ) * sqrt ( q1 * q / h );
                        K1[j].A=h*dy[j].A;//+h*gv(0,T,G)*w1;
                        w1 = Normal(0, sqrt(2*Kb*T*m*G)) * sqrt ( q1 * q / h );
                        K1[j].TH=h*dy[j].TH+h*w1;//+h*gv(0,T,G)*w1;
                        //w1 = r8_normal_01 (seed ) * sqrt ( q1 * q / h );
                        K1[j].OM=h*dy[j].OM;//+h*gv(0,T,G)*w1;
```

```
                        //w1 = r8_normal_01 (seed ) * sqrt ( q1 * q / h );
                        K1[j].XR=h*dy[j].XR;//+h*gv(0,T,G)*w1;
                        //w1 = r8_normal_01 (seed ) * sqrt ( q1 * q / h );
                        K1[j].XI=h*dy[j].XI;//+h*gv(0,T,G)*w1;
                        //w1 = r8_normal_01 (seed ) * sqrt ( q1 * q / h );
                        K1[j].HR=h*dy[j].HR;//+h*gv(0,T,G)*w1;
                        //w1 = r8_normal_01 (seed ) * sqrt ( q1 * q / h );
                        K1[j].HI=h*dy[j].HI;//+h*gv(0,T,G)*w1;

                }
/*****************************************************************************/

/*****************************************************************************/

        t2=t1+a21*h;
        for (j=ghost;j<M+ghost;j++)
                {
                        yk[j].Y=BP[j].Y+a21*K1[j].Y;
                        yk[j].A=BP[j].A+a21*K1[j].A;
                        yk[j].TH=BP[j].TH+a21*K1[j].TH;
                        OLD[j]=yk[j].OM;
                        yk[j].OM=BP[j].OM+a21*K1[j].OM;
                        yk[j].XR=BP[j].XR+a21*K1[j].XR;
                        yk[j].XI=BP[j].XI+a21*K1[j].XI;
                        yk[j].HR=BP[j].HR+a21*K1[j].HR;
                        yk[j].HI=BP[j].HI+a21*K1[j].HI;
                }
        Distribute_data(yk,5,6,Job_ID);
        RHS_FUN(yk,dy,Strand,X,V,G,OLD,h,Total_Energy,Transport_Study);


        for (j=ghost;j<M+ghost;j++)
                {
                  w2 = Normal(0, sqrt(2*Kb*T*m*G))* sqrt ( q2 * q / h );
                  K2[j].Y=h*dy[j].Y+h*w2;
                  //w2 = r8_normal_01 (seed ) * sqrt ( q2 * q / h );
                  K2[j].A=h*dy[j].A;//+h*gv(0,T,G)*w2;
                  w2 = Normal(0, sqrt(2*Kb*T*m*G))* sqrt ( q2 * q / h );
                  K2[j].TH=h*dy[j].TH+h*w2;//+h*gv(0,T,G)*w2;
                  //w2 = r8_normal_01 (seed ) * sqrt ( q2 * q / h );
                  K2[j].OM=h*dy[j].OM;//+h*gv(0,T,G)*w2;
                  //w2 = r8_normal_01 (seed ) * sqrt ( q2 * q / h );
                  K2[j].XR=h*dy[j].XR;//+h*gv(0,T,G)*w2;
                  //w2 = r8_normal_01 (seed ) * sqrt ( q2 * q / h );
                  K2[j].XI=h*dy[j].XI;//+h*gv(0,T,G)*w2;
                  //w2 = r8_normal_01 (seed ) * sqrt ( q2 * q / h );
                  K2[j].HR=h*dy[j].HR;//+h*gv(0,T,G)*w2;
                  //w2 = r8_normal_01 (seed ) * sqrt ( q2 * q / h );
                  K2[j].HI=h*dy[j].HI;//+h*gv(0,T,G)*w2;


                }
/*****************************************************************************/

/*****************************************************************************/
```

```
t3=t1+a31*h+a32*h;
for (j=ghost;j<M+ghost;j++)
        {
                yk[j].Y=BP[j].Y+a31*K1[j].Y+a32*K2[j].Y;
                yk[j].A=BP[j].A+a31*K1[j].A+a32*K2[j].A;
                yk[j].TH=BP[j].TH+a31*K1[j].TH+a32*K2[j].TH;
                OLD[j]=yk[j].OM;
                yk[j].OM=BP[j].OM+a31*K1[j].OM+a32*K2[j].OM;
                yk[j].XR=BP[j].XR+a31*K1[j].XR+a32*K2[j].XR;
                yk[j].XI=BP[j].XI+a31*K1[j].XI+a32*K2[j].XI;
                yk[j].HR=BP[j].HR+a31*K1[j].HR+a32*K2[j].HR;
                yk[j].HI=BP[j].HI+a31*K1[j].HI+a32*K2[j].HI;
        }

Distribute_data(yk,7,8,Job_ID);
RHS_FUN(yk,dy,Strand,X,V,G,OLD,h,Total_Energy,Transport_Study);


for (j=ghost;j<M+ghost;j++)
        {
                w3 = Normal(0, sqrt(2*Kb*T*m*G)) * sqrt ( q3 * q / h );
                K3[j].Y=h*dy[j].Y+h*w3;
                //w3 = r8_normal_01 (seed ) * sqrt ( q3 * q / h );
                K3[j].A=h*dy[j].A;//+h*gv(0,T,G)*w3;
                w3 = Normal(0, sqrt(2*Kb*T*m*G)) * sqrt ( q3 * q / h );
                K3[j].TH=h*dy[j].TH+h*w3;//+h*gv(0,T,G)*w3;
                //w3 = r8_normal_01 (seed ) * sqrt ( q3 * q / h );
                K3[j].OM=h*dy[j].OM;//+h*gv(0,T,G)*w3;
                //w3 = r8_normal_01 (seed ) * sqrt ( q3 * q / h );
                K3[j].XR=h*dy[j].XR;//+h*gv(0,T,G)*w3;
                //w3 = r8_normal_01 (seed ) * sqrt ( q3 * q / h );
                K3[j].XI=h*dy[j].XI;//+h*gv(0,T,G)*w3;
                //w3 = r8_normal_01 (seed ) * sqrt ( q3 * q / h );
                K3[j].HR=h*dy[j].HR;//+h*gv(0,T,G)*w3;
                //w3 = r8_normal_01 (seed ) * sqrt ( q3 * q / h );
                K3[j].HI=h*dy[j].HI;//+h*gv(0,T,G)*w3;
        }
/*************************************************************************/

/*************************************************************************/

t4=t1+a41*h+a42*h+a43*h;
for (j=ghost;j<M+ghost;j++)
        {
                yk[j].Y=BP[j].Y+a41*K1[j].Y+a42*K2[j].Y+a43*K3[j].Y;
                yk[j].A=BP[j].A+a41*K1[j].A+a42*K2[j].A+a43*K3[j].A;
                yk[j].TH=BP[j].TH+a41*K1[j].TH+a42*K2[j].TH+a43*K3[j].TH;
                OLD[j]=yk[j].OM;
                yk[j].OM=BP[j].OM+a41*K1[j].OM+a42*K2[j].OM+a43*K3[j].OM;
                yk[j].XR=BP[j].XR+a41*K1[j].XR+a42*K2[j].XR+a43*K3[j].XR;
                yk[j].XI=BP[j].XI+a41*K1[j].XI+a42*K2[j].XI+a43*K3[j].XI;
                yk[j].HR=BP[j].HR+a41*K1[j].HR+a42*K2[j].HR+a43*K3[j].HR;
                yk[j].HI=BP[j].HI+a41*K1[j].HI+a42*K2[j].HI+a43*K3[j].HI;

        }
 Distribute_data(yk,9,10,Job_ID);
```

```
        RHS_FUN(yk,dy,Strand,X,V,G,OLD,h,Total_Energy,Transport_Study);


      for (j=ghost;j<M+ghost;j++)
            {
                  w4 = Normal(0, sqrt(2*Kb*T*m*G)) * sqrt ( q4 * q / h );
                  K4[j].Y=h*dy[j].Y+h*w4;
                  //w4 = r8_normal_01 ( seed ) * sqrt ( q4 * q / h );
                  K4[j].A=h*dy[j].A;//+h*gv(0,T,G)*w4;
                  w4 = Normal(0, sqrt(2*Kb*T*m*G)) * sqrt ( q4 * q / h );
                  K4[j].TH=h*dy[j].TH+h*w4;//+h*gv(0,T,G)*w4;
                  //w4 = r8_normal_01 ( seed ) * sqrt ( q4 * q / h );
                  K4[j].OM=h*dy[j].OM;//+h*gv(0,T,G)*w4;
                  //w4 = r8_normal_01 ( seed ) * sqrt ( q4 * q / h );
                  K4[j].XR=h*dy[j].XR;//+h*gv(0,T,G)*w4;
                  //w4 = r8_normal_01 ( seed ) * sqrt ( q4 * q / h );
                  K4[j].XI=h*dy[j].XI;//+h*gv(0,T,G)*w4;
                  //w4 = r8_normal_01 ( seed ) * sqrt ( q4 * q / h );
                  K4[j].HR=h*dy[j].HR;//+h*gv(0,T,G)*w4;
                  //w4 = r8_normal_01 ( seed ) * sqrt ( q4 * q / h );
                  K4[j].HI=h*dy[j].HI;//+h*gv(0,T,G)*w4;
            }
/*****************************************************************************/

/*****************************************************************************/
   for (j=ghost;j<M+ghost;j++)
            {
                  BP[j].Y = BP[j].Y + a51 * K1[j].Y + a52 * K2[j].Y + a53 * K3[j].Y + a54 * K4[j].Y;
                  BP[j].A = BP[j].A + a51 * K1[j].A + a52 * K2[j].A + a53 * K3[j].A + a54 * K4[j].A;
                  BP[j].TH = BP[j].TH + a51 * K1[j].TH + a52 * K2[j].TH + a53 * K3[j].TH + a54 *
K4[j].TH;
                  BP[j].OM = BP[j].OM + a51 * K1[j].OM + a52 * K2[j].OM + a53 * K3[j].OM + a54 *
K4[j].OM;
                  BP[j].XR = BP[j].XR + a51 * K1[j].XR + a52 * K2[j].XR + a53 * K3[j].XR + a54 *
K4[j].XR;
                  BP[j].XI = BP[j].XI + a51 * K1[j].XI + a52 * K2[j].XI + a53 * K3[j].XI + a54 *
K4[j].XI;
                  BP[j].HR = BP[j].HR + a51 * K1[j].HR + a52 * K2[j].HR + a53 * K3[j].HR + a54 *
K4[j].HR;
                  BP[j].HI = BP[j].HI + a51 * K1[j].HI + a52 * K2[j].HI + a53 * K3[j].HI + a54 *
K4[j].HI;
            }

/*****************************************************************************/


free(OLD);
 OLD=NULL;
free(dy);
 dy=NULL;
free(yk);
yk=NULL;
free(K1);
 K1=NULL;
free(K2);
 K2=NULL;
```

```
free(K3);
K3=NULL;
free(K4);
K4=NULL;

}


/**************************************************************************/

double gv ( int x ,int T,double gma)

/**************************************************************************/
/*
  Purpose:

    GI is a time invariant stochastic right hand side.

  Licensing:

    This code is distributed under the GNU LGPL license.

  Modified:

    07 July 2010

  Author:

    John Burkardt

  Parameters:

    Input, double X, the argument.

    Output, double GI, the value.
*/
{

  double value,Kb,m;
  Kb=8.617343e-5;
  //gma=.005;
  m=300*1.0358e-4;
  value=0;
  if (x==1)
  {

  value = sqrt(2*Kb*T*m*gma);
  }
  else
  {
            value=0.0;;
  }
  return value;
}
```

(v) RHS function for fourth order Runge-Kutta method

```
/*****************************************************************************/
/*
  Purpose:
    RHS function for fourth order Runge-Kutta method
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
    07 July 2010
  Author:
          Neranjan Edirisinghe
*/
/*****************************************************************************/

# include "stochastic_rk.h"

void RHS_FUN(Base_Pair yk[],Base_Pair dy[], Base Strand[],double X,double V,double GM,double OLD[],double
STEP,double Total_Energy[],int Transport_Study)
{

          int i,j,C1,C2,M,size,myrank;
          double
Vfun1,Wfun12,Wfun21,a,h,m,k,R,B,D,R0,L,ld,R0I,THA,KI,Kyy,Kpp,Kyp,G,GI,H,G0,K,td,Xe,Xh,X3,V1,V2,XI1,XI3,X
R1,XR3;
          double
DCHANGE,HDASH,TH2,OM2,XI2,XR2,HI2,HR2,YT1,YT2,YT3,TEMP1,TEMP2,TEMP3,TEMP4,TEMP5,TEMP6,T
EMP7,TEMP8,X1,X2,H2,H1,DR,TH1,TH3;
          //Base_Pair *BP;
           MPI_Comm_size ( MPI_COMM_WORLD, &size );
          MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
          M=N/size;
          Xe=X;
          Xh=X;
          C1=1;
          C2=2;
          a=4.45;
          D=0.04;
          k=0.4;
          B=0.35;
          R=0.5;
          h=0.000658211;
          m=0.03105;
          HDASH=sqrt(pow(a,2)/(D*m));
          KI=1;
          R0I=10;
          THA=25;
          H=3.4;
          DR=a*3.4;
          ld=sqrt(pow(H,2)+4*pow(R0I,2)*pow(sin(THA/2),2));
          R0=a*R0I;
          td=sqrt(D*pow(a,2)/m);
          h=h*HDASH;
          //STEP=STEP*td;
          G0=KI*pow(R0,2)/2;
          L=a*ld;
          GI=G0/D;
```

```
        K=KI/(D*pow(a,2));
        DCHANGE=1/(D*a);

        Kyy=(K*pow(R0,2)/pow(L,2))*(pow((1-cos(THA)),2));
        Kpp=(K*pow(R0,2)/pow(L,2))*pow(sin(THA),2);
        Kyp=2*(K*pow(R0,2)/pow(L,2))*sin(THA)*(1-cos(THA));
        G=GI/pow(R0,2);
        Vfun1=0;
        Wfun12=0;
        Wfun21=0;



for (j=ghost;j<M+ghost;j++)
{

        YT2=yk[j].Y;
   YT1=yk[j-1].Y;
   YT3=yk[j+1].Y;
   XR2=yk[j].XR;
   XI2=yk[j].XI;
   XR1=yk[j-1].XR;
   XI1=yk[j-1].XI;
   XR3=yk[j+1].XR;
   XI3=yk[j+1].XI;
   HR2=yk[j].HR;
   HI2=yk[j].HI;
   TH1=yk[j-1].TH;
   TH2=yk[j].TH;
   TH3=yk[j+1].TH;
   OM2=yk[j].OM;


   TEMP1=0;
   TEMP2=0;
   TEMP3=0;
   TEMP4=0;
   TEMP5=0;
   TEMP6=0;
   TEMP7=0;
   TEMP8=0;

for (i=1;i<ghost;i++)
        {


                H2=(pow(yk[j+i].HR,2)+pow(yk[j+i].HI,2));
                X2=(pow(yk[j+i].XR,2)+pow(yk[j+i].XI,2));
                H1=(pow(yk[(j-i)].HR,2)+pow(yk[(j-i)].HI,2));
                X1=(pow(yk[(j-i)].XR,2)+pow(yk[(j-i)].XI,2));
                TEMP1=TEMP1+(XI2*H2+XI2*H1)/((i)*DR);
                TEMP2=TEMP2+(XR2*H2+XR2*H1)/((i)*DR);
                TEMP3=TEMP3+(HI2*X2+HI2*X1)/((i)*DR);
                TEMP4=TEMP4+(HR2*X2+HR2*X1)/((i)*DR);
                TEMP5=TEMP5+(HI2*H2+HI2*H1)/((i)*DR);
```

```
                                TEMP6=TEMP6+(HR2*H2+HR2*H1)/((i)*DR);
                                TEMP7=TEMP7+(XI2*X2+XI2*X1)/((i)*DR);
                                TEMP8=TEMP8+(XR2*X2+XR2*X1)/((i)*DR);


        }
   H2=(pow(HR2,2)+pow(HI2,2));
         X3=(pow(XR3,2)+pow(XI3,2));
         X1=(pow(XR1,2)+pow(XI1,2));
         X2=(pow(XR2,2)+pow(XI2,2));
   V1=(1/D)*V_cal(TH2,YT2,TH3,YT3,j,Transport_Study);
         V2=(1/D)*V_cal(TH1,YT1,TH2,YT2,j-1,Transport_Study);
        dy[j].HR=Total_Energy[j]*HR2+(-V1*yk[1+j].HI-V2*yk[-1+j].HI-
(Xh*DCHANGE)*YT2*HI2+Strand[j].well_depth*HI2)/h/*-Recombiantion_Rate*HR2*X2*/+TEMP1/h-
TEMP5/h+GRADIENT*j*yk[j].HR-td*Strand[j].absorption*HR2;
        dy[j].HI=Total_Energy[j]*HI2+(V1*yk[1+j].HR+V2*yk[-1+j].HR+(Xh*DCHANGE)*YT2*HR2-
Strand[j].well_depth*HR2)/h/*-Recombiantion_Rate*HI2*X2*/-TEMP2/h+TEMP6/h+GRADIENT*j*yk[j].HI-
td*Strand[j].absorption*HI2;
        dy[j].XR=Total_Energy[j]*XR2+(-V1*XI3-
V2*XI1+(Xe*DCHANGE)*YT2*XI2+Strand[j].well_depth*XI2)/h/*-Recombiantion_Rate*XR2*H2+TEMP3/h-
TEMP7/h-GRADIENT*j*yk[j].XR*/-td*Strand[j].absorption*XR2;
        dy[j].XI=Total_Energy[j]*XI2+(V1*XR3+V2*XR1-(Xe*DCHANGE)*YT2*XR2-
Strand[j].well_depth*XR2)/h/*-Recombiantion_Rate*XI2*H2-TEMP4/h+TEMP8/h-GRADIENT*j*yk[j].XI*/-
td*Strand[j].absorption*XI2;
        dy[j].A=(1+YT2/R0)*(1/R0)*pow(OM2,2)-(YT2-(3/2)*pow(YT2,2)+(7/6)*pow(YT2,3))-
Kyy*(YT3+YT1+2*YT2)-Kyp*(TH3-TH1)-(Xe*X2-Xh*H2)*DCHANGE;
        dy[j].Y=yk[j].A;
         dy[j].OM=Kpp*(TH3+TH1-2*TH2)-G*(yk[j+2].TH+yk[j-2].TH-4*TH3-4*TH1+6*TH2)+(Kyp/2)*(YT3-
YT1)-(2/R0)*yk[j].A*OM2-(2/R0)*YT2*((OM2-OLD[j])/STEP)-(2/pow(R0,2))*YT2*yk[j].A*OM2-
(1/pow(R0,2))*pow(YT2,2)*((OM2-OLD[j])/STEP)+V1*TH2*(X2*X1)/D+V2*TH2*(X3*X2)/D;
                        dy[j].TH=OM2;


}


}
```

(vi) Runge-Kutta-Fehlberg method

```
/******************************************************************************/
/*
  Purpose:
    Runge-Kutta-Fehlberg method
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
    07 July 2010
  Author:
        Neranjan Edirisinghe
*/
/******************************************************************************/
```

```
# include "stochastic_rk.h"
void Runge_Kutta_Fehlberg( Base_Pair BP[],Base_Pair y2[],Base_Pair y1[],Base_Pair y0[],double h,double t,
Base Strand[],double X,double V,int T,double G,double Total_Energy[],int Transport_Study,int Job_ID)
{
  //double h;
  int j;
  //int n;
  double q,Multiplier,*OLD;
  int seed;
    int myrank,size,M,Send_count_start;
            int   Destination,Origin;
            MPI_Status M_status;
              Base_Pair *yk,*ytemp1,*ytemp2,*dy;
   q = 1.0;
  seed = 123456789;

            MPI_Comm_size ( MPI_COMM_WORLD, &size );
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
M=N/size;
Send_count_start=M;

                        yk=malloc ((M+2*ghost)*sizeof(Base_Pair));
                        if (yk == NULL)
                        {
                                printf("error defining yk  ");
                        }

                        ytemp1=malloc ((M+2*ghost)*sizeof(Base_Pair));
                        if (ytemp1 == NULL)
                        {
                                printf("error defining ytemp1  ");
                        }

                        ytemp2=malloc ((M+2*ghost)*sizeof(Base_Pair));
                        if (ytemp2 == NULL)
                        {
                                printf("error defining ytemp2  ");
                        }
                        OLD=malloc ((M+2*ghost)*sizeof(double));
                        if (OLD == NULL)
                        {
                                printf("error defining OLD  ");
                        }
                        dy=malloc ((M+2*ghost)*sizeof(Base_Pair));
                        if (dy == NULL)
                        {
                                printf("error defining dy  ");
                        }


                                for (j=0;j<(M+2*ghost);j++)
                                {

                                        yk[j].HR=BP[j].HR;
                                        yk[j].XR=BP[j].XR;
                                        yk[j].HI=BP[j].HI;
```

```
                yk[j].XI=BP[j].XI;
                yk[j].Y=BP[j].Y;
                yk[j].A=BP[j].A;
                yk[j].OM=BP[j].OM;
                yk[j].TH=BP[j].TH;
                OLD[j]=y2[j].OM;
                ytemp1[j].HR=0;
                ytemp1[j].XR=0;
                ytemp1[j].HI=0;
                ytemp1[j].XI=0;
                ytemp1[j].Y=0;
                ytemp1[j].A=0;
                ytemp1[j].OM=0;
                ytemp1[j].TH=0;
                ytemp2[j].HR=0;
                ytemp2[j].XR=0;
                ytemp2[j].HI=0;
                ytemp2[j].XI=0;
                ytemp2[j].Y=0;
                ytemp2[j].A=0;
                ytemp2[j].OM=0;
                ytemp2[j].TH=0;



        }


RHS_FUN(yk,dy,Strand,X,V,G,OLD,h,Total_Energy,Transport_Study);
Multiplier=55*h;
        for (j=0;j<(M+2*ghost);j++)
        {

                ytemp1[j].HR=ytemp1[j].HR+Multiplier*dy[j].HR;
                ytemp1[j].XR=ytemp1[j].XR+Multiplier*dy[j].XR;
                ytemp1[j].HI=ytemp1[j].HI+Multiplier*dy[j].HI;
                ytemp1[j].XI=ytemp1[j].XI+Multiplier*dy[j].XI;
                ytemp1[j].Y=ytemp1[j].Y+Multiplier*dy[j].Y;
                ytemp1[j].A=ytemp1[j].A+Multiplier*dy[j].A;
                ytemp1[j].OM=ytemp1[j].OM+Multiplier*dy[j].OM;
                ytemp1[j].TH=ytemp1[j].TH+Multiplier*dy[j].TH;
                OLD[j]=y2[j].OM;



        }

        Multiplier=19*h;
        for (j=0;j<(M+2*ghost);j++)
        {

                ytemp2[j].HR=ytemp2[j].HR+Multiplier*dy[j].HR;
                ytemp2[j].XR=ytemp2[j].XR+Multiplier*dy[j].XR;
                ytemp2[j].HI=ytemp2[j].HI+Multiplier*dy[j].HI;
                ytemp2[j].XI=ytemp2[j].XI+Multiplier*dy[j].XI;
                ytemp2[j].Y=ytemp2[j].Y+Multiplier*dy[j].Y;
```

```
                ytemp2[j].A=ytemp2[j].A+Multiplier*dy[j].A;
                ytemp2[j].OM=ytemp2[j].OM+Multiplier*dy[j].OM;
                ytemp2[j].TH=ytemp2[j].TH+Multiplier*dy[j].TH;


        }


    for (j=0;j<(M+2*ghost);j++)
        {

                yk[j].HR=y2[j].HR;
                yk[j].XR=y2[j].XR;
                yk[j].HI=y2[j].HI;
                yk[j].XI=y2[j].XI;
                yk[j].Y=y2[j].Y;
                yk[j].A=y2[j].A;
                yk[j].OM=y2[j].OM;
                yk[j].TH=y2[j].TH;
                OLD[j]=y1[j].OM;



        }

RHS_FUN(yk,dy,Strand,X,V,G,OLD,h,Total_Energy,Transport_Study);
Multiplier=-59*h;
        for (j=0;j<(M+2*ghost);j++)
        {

                ytemp1[j].HR=ytemp1[j].HR+Multiplier*dy[j].HR;
                ytemp1[j].XR=ytemp1[j].XR+Multiplier*dy[j].XR;
                ytemp1[j].HI=ytemp1[j].HI+Multiplier*dy[j].HI;
                ytemp1[j].XI=ytemp1[j].XI+Multiplier*dy[j].XI;
                ytemp1[j].Y=ytemp1[j].Y+Multiplier*dy[j].Y;
                ytemp1[j].A=ytemp1[j].A+Multiplier*dy[j].A;
                ytemp1[j].OM=ytemp1[j].OM+Multiplier*dy[j].OM;
                ytemp1[j].TH=ytemp1[j].TH+Multiplier*dy[j].TH;




        }


        Multiplier=-5*h;
        for (j=0;j<(M+2*ghost);j++)
        {

                ytemp2[j].HR=ytemp2[j].HR+Multiplier*dy[j].HR;
                ytemp2[j].XR=ytemp2[j].XR+Multiplier*dy[j].XR;
                ytemp2[j].HI=ytemp2[j].HI+Multiplier*dy[j].HI;
                ytemp2[j].XI=ytemp2[j].XI+Multiplier*dy[j].XI;
                ytemp2[j].Y=ytemp2[j].Y+Multiplier*dy[j].Y;
                ytemp2[j].A=ytemp2[j].A+Multiplier*dy[j].A;
```

```
                ytemp2[j].OM=ytemp2[j].OM+Multiplier*dy[j].OM;
                ytemp2[j].TH=ytemp2[j].TH+Multiplier*dy[j].TH;


        }

        for (j=0;j<(M+2*ghost);j++)
        {

                yk[j].HR=y1[j].HR;
                yk[j].XR=y1[j].XR;
                yk[j].HI=y1[j].HI;
                yk[j].XI=y1[j].XI;
                yk[j].Y=y1[j].Y;
                yk[j].A=y1[j].A;
                yk[j].OM=y1[j].OM;
                yk[j].TH=y1[j].TH;
                OLD[j]=y0[j].OM;



        }

RHS_FUN(yk,dy,Strand,X,V,G,OLD,h,Total_Energy,Transport_Study);
Multiplier=37*h;
        for (j=0;j<(M+2*ghost);j++)
        {

                ytemp1[j].HR=ytemp1[j].HR+Multiplier*dy[j].HR;
                ytemp1[j].XR=ytemp1[j].XR+Multiplier*dy[j].XR;
                ytemp1[j].HI=ytemp1[j].HI+Multiplier*dy[j].HI;
                ytemp1[j].XI=ytemp1[j].XI+Multiplier*dy[j].XI;
                ytemp1[j].Y=ytemp1[j].Y+Multiplier*dy[j].Y;
                ytemp1[j].A=ytemp1[j].A+Multiplier*dy[j].A;
                ytemp1[j].OM=ytemp1[j].OM+Multiplier*dy[j].OM;
                ytemp1[j].TH=ytemp1[j].TH+Multiplier*dy[j].TH;


        }

        Multiplier=h;
        for (j=0;j<(M+2*ghost);j++)
        {

                ytemp2[j].HR=ytemp2[j].HR+Multiplier*dy[j].HR;
                ytemp2[j].XR=ytemp2[j].XR+Multiplier*dy[j].XR;
                ytemp2[j].HI=ytemp2[j].HI+Multiplier*dy[j].HI;
                ytemp2[j].XI=ytemp2[j].XI+Multiplier*dy[j].XI;
                ytemp2[j].Y=ytemp2[j].Y+Multiplier*dy[j].Y;
                ytemp2[j].A=ytemp2[j].A+Multiplier*dy[j].A;
                ytemp2[j].OM=ytemp2[j].OM+Multiplier*dy[j].OM;
                ytemp2[j].TH=ytemp2[j].TH+Multiplier*dy[j].TH;


        }
```

```
                for (j=0;j<(M+2*ghost);j++)
                {

                        yk[j].HR=y0[j].HR;
                        yk[j].XR=y0[j].XR;
                        yk[j].HI=y0[j].HI;
                        yk[j].XI=y0[j].XI;
                        yk[j].Y=y0[j].Y;
                        yk[j].A=y0[j].A;
                        yk[j].OM=y0[j].OM;
                        yk[j].TH=y0[j].TH;
                        OLD[j]=y0[j].OM;



                }

RHS_FUN(yk,dy,Strand,X,V,G,OLD,h,Total_Energy,Transport_Study);
Multiplier=-9*h;
                for (j=0;j<(M+2*ghost);j++)
                {

                        ytemp1[j].HR=ytemp1[j].HR+Multiplier*dy[j].HR;
                        ytemp1[j].XR=ytemp1[j].XR+Multiplier*dy[j].XR;
                        ytemp1[j].HI=ytemp1[j].HI+Multiplier*dy[j].HI;
                        ytemp1[j].XI=ytemp1[j].XI+Multiplier*dy[j].XI;
                        ytemp1[j].Y=ytemp1[j].Y+Multiplier*dy[j].Y;
                        ytemp1[j].A=ytemp1[j].A+Multiplier*dy[j].A;
                        ytemp1[j].OM=ytemp1[j].OM+Multiplier*dy[j].OM;
                        ytemp1[j].TH=ytemp1[j].TH+Multiplier*dy[j].TH;


                }

                for (j=0;j<(M+2*ghost);j++)
                {

                        ytemp1[j].HR=ytemp1[j].HR/24+BP[j].HR;
                        ytemp1[j].XR=ytemp1[j].XR/24+BP[j].XR;
                        ytemp1[j].HI=ytemp1[j].HI/24+BP[j].HI;
                        ytemp1[j].XI=ytemp1[j].XI/24+BP[j].XI;
                        ytemp1[j].Y=ytemp1[j].Y/24+BP[j].Y;
                        ytemp1[j].A=ytemp1[j].A/24+BP[j].A;
                        ytemp1[j].OM=ytemp1[j].OM/24+BP[j].OM;
                        ytemp1[j].TH=ytemp1[j].TH/24+BP[j].TH;


                }

                for (j=0;j<(M+2*ghost);j++)
                {

                        yk[j].HR=ytemp1[j].HR;
                        yk[j].XR=ytemp1[j].XR;
                        yk[j].HI=ytemp1[j].HI;
                        yk[j].XI=ytemp1[j].XI;
```

```
                    yk[j].Y=ytemp1[j].Y;
                    yk[j].A=ytemp1[j].A;
                    yk[j].OM=ytemp1[j].OM;
                    yk[j].TH=ytemp1[j].TH;
                    OLD[j]=BP[j].OM;



                }

        Send_count_start=M;
         Destination=myrank+1;
         if (Destination>size-1)
         {//{3}
                Destination=0;
         }//{3}

         Origin=myrank-1;
         if (Origin<0)
         {//{3}
                 Origin=size-1;
         }//{3}



        MPI_Sendrecv( &yk[Send_count_start], DIM*ghost,MPI_DOUBLE,Destination,4,&yk[0],
DIM*ghost,MPI_DOUBLE, Origin,4,MPI_COMM_WORLD, &M_status);
         Send_count_start=ghost;
        Destination=myrank-1;
         if (Destination<0)
         {//{3}
                Destination=size-1;
         }//{3}

         Origin=myrank+1;
         if (Origin>size-1)
         {//{3}
                 Origin=0;
         }//{3}

   MPI_Sendrecv(&yk[Send_count_start],DIM*ghost,MPI_DOUBLE,Destination,3, &yk[M+ghost],
DIM*ghost,MPI_DOUBLE, Origin,3,MPI_COMM_WORLD, &M_status);
   MPI_Barrier(MPI_COMM_WORLD);


                    RHS_FUN(yk,dy,Strand,X,V,G,OLD,h,Total_Energy,Transport_Study);
                    Multiplier=9*h;
                    for (j=0;j<(M+2*ghost);j++)
                    {

                        ytemp2[j].HR=ytemp2[j].HR+Multiplier*dy[j].HR;
                        ytemp2[j].XR=ytemp2[j].XR+Multiplier*dy[j].XR;
                        ytemp2[j].HI=ytemp2[j].HI+Multiplier*dy[j].HI;
                        ytemp2[j].XI=ytemp2[j].XI+Multiplier*dy[j].XI;
                        ytemp2[j].Y=ytemp2[j].Y+Multiplier*dy[j].Y;
                        ytemp2[j].A=ytemp2[j].A+Multiplier*dy[j].A;
```

```
            ytemp2[j].OM=ytemp2[j].OM+Multiplier*dy[j].OM;
            ytemp2[j].TH=ytemp2[j].TH+Multiplier*dy[j].TH;


}

for (j=0;j<(M+2*ghost);j++)
{

            ytemp1[j].HR=ytemp2[j].HR/24+BP[j].HR;
            ytemp1[j].XR=ytemp2[j].XR/24+BP[j].XR;
            ytemp1[j].HI=ytemp2[j].HI/24+BP[j].HI;
            ytemp1[j].XI=ytemp2[j].XI/24+BP[j].XI;
            ytemp1[j].Y=ytemp2[j].Y/24+BP[j].Y;
            ytemp1[j].A=ytemp2[j].A/24+BP[j].A;
            ytemp1[j].OM=ytemp2[j].OM/24+BP[j].OM;
            ytemp1[j].TH=ytemp2[j].TH/24+BP[j].TH;


}


for (j=0;j<(M+2*ghost);j++)
{

            y0[j].HR=y1[j].HR;
            y0[j].XR=y1[j].XR;
            y0[j].HI=y1[j].HI;
            y0[j].XI=y1[j].XI;
            y0[j].Y=y1[j].Y;
            y0[j].A=y1[j].A;
            y0[j].OM=y1[j].OM;
            y0[j].TH=y1[j].TH;


}


for (j=0;j<(M+2*ghost);j++)
{

            y1[j].HR=y2[j].HR;
            y1[j].XR=y2[j].XR;
            y1[j].HI=y2[j].HI;
            y1[j].XI=y2[j].XI;
            y1[j].Y=y2[j].Y;
            y1[j].A=y2[j].A;
            y1[j].OM=y2[j].OM;
            y1[j].TH=y2[j].TH;


}

for (j=0;j<(M+2*ghost);j++)
{
```

```
                    y2[j].HR=BP[j].HR;
                    y2[j].XR=BP[j].XR;
                    y2[j].HI=BP[j].HI;
                    y2[j].XI=BP[j].XI;
                    y2[j].Y=BP[j].Y;
                    y2[j].A=BP[j].A;
                    y2[j].OM=BP[j].OM;
                    y2[j].TH=BP[j].TH;


                }

                for (j=0;j<(M+2*ghost);j++)
                {

                    BP[j].HR=ytemp1[j].HR;
                    BP[j].XR=ytemp1[j].XR;
                    BP[j].HI=ytemp1[j].HI;
                    BP[j].XI=ytemp1[j].XI;
                    BP[j].Y=ytemp1[j].Y;
                    BP[j].A=ytemp1[j].A;
                    BP[j].OM=ytemp1[j].OM;
                    BP[j].TH=ytemp1[j].TH;


                }

                     Send_count_start=M;
        Destination=myrank+1;
        if (Destination>size-1)
        {//{3}
            Destination=0;
        }//{3}

        Origin=myrank-1;
        if (Origin<0)
        {//{3}
             Origin=size-1;
        }//{3}



      MPI_Sendrecv( &BP[Send_count_start], DIM*ghost,MPI_DOUBLE,Destination,4,&BP[0],
DIM*ghost,MPI_DOUBLE, Origin,4,MPI_COMM_WORLD, &M_status);

        Send_count_start=ghost;
        Destination=myrank-1;
        if (Destination<0)
        {//{3}
            Destination=size-1;
        }//{3}

        Origin=myrank+1;
        if (Origin>size-1)
        {//{3}
             Origin=0;
```

```
        }//{3}

   MPI_Sendrecv(&BP[Send_count_start],DIM*ghost,MPI_DOUBLE,Destination,3, &BP[M+ghost],
DIM*ghost,MPI_DOUBLE, Origin,3,MPI_COMM_WORLD, &M_status);
   MPI_Barrier(MPI_COMM_WORLD);

free(OLD);
OLD=NULL;
free(dy);
dy=NULL;
free(yk);
yk=NULL;
free(ytemp1);
ytemp1=NULL;
free(ytemp2);
ytemp2=NULL;
  return;
}
```

(vii) Header file

```
/*****************************************************************************/
/*
  Purpose:
    Header file
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
    07 July 2010
  Author:
          Neranjan Edirisinghe
*/
/*****************************************************************************/

#include "mpi.h"
#include "math.h"
#include "stdio.h"
#include "malloc.h"
# include <time.h>
#include <string.h>
# include <stdio.h>
# include <stdlib.h>
# include <math.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include "rngs.h"
#include "rvgs.h"
#define DIM 8
#define DEBUG 0


#define Eg 7.77
#define Ec 8.87
```

```
#define Ea 8.25
#define Et 9.13
#define tb 1.5
#define tw 0.15

#define T2 tb*tb
#define Alpha1 (Eg*Eg+Ec*Ec)/T2
#define E_Init 2*(Eg+Ec)
typedef struct
        {
      double HR;
      double HI;
                  double XR;
      double XI;
                  double TH;
                  double OM;
      double Y;
      double A;

        } Base_Pair;

typedef struct
        {
      double Wave_Function;
      double Displacement;
                  double Twist_Angle;

        } Storage_Base_Pair;

typedef struct
        {
      int Ray_Hit;
      int Position;
                  int Sub_unit;

        } Radiation;



typedef struct
        {
        int Initiate;
        int Sub_unit;
        int Position;

} Cell;

typedef struct
        {
        double well_depth;
        double absorption;
} Base;


typedef struct
        {
```

```
        double Return_Probability;
        double Participation_Function;
        double Shannon_Information_Entrophy;
} Norm;

#define N 512
#define ghost 5
#define Recombiantion_Rate 200
#define  GRADIENT 0.0001

//extern double r8_normal_01 ( int *seed );
//extern double r8_uniform_01 ( int *seed );
//void timestamp ( );
extern void RHS_FUN(Base_Pair yk[],Base_Pair dy[], Base Strand[],double X,double V,double G,double
OLD[],double,double Total_Energy[],int Transport_Study);
extern void E_THO_CONST(double E[],double THO[],int QD,double QW1, double THO1);
extern void Stochastic_RK4(Base_Pair BP[], Base Strand[],double X,double V,double t,double h,double q,int *seed,
int T,double G,double Total_Energy[],int Transport_Study,int);
//extern double r8_normal_01 ( int *seed );
extern double Total_Energy(double y[]);
extern double V_cal(double theta1, double r1,double theta2, double r2, int i,int Transport_Study);
extern double V_fun(double a0,double a1, double a2,double a3,double a4,double b1,double b2,double b3,double
b4,double w,double x,int FI);
extern void Print_matrix(Base_Pair BP_MAIN[],FILE *yfile,FILE *afile,FILE *tfile,FILE *ofile,FILE *wfile,FILE
*hfile);
extern void Distribute_data(Base_Pair yk[],int,int,int);
extern void Onsite_Energy_Calculation(Base_Pair BP[],double Total_Energy[],double X,double
Hydration_Energy,int Transport_Study);
extern void Energy_Calculation(Base_Pair BP[],double Total_Energy[],double X,int Transport_Study,int);
extern double Normal(double m, double s);
extern void PutSeed(long x);
extern double Maxwell_Boltzman(double G,int T,int);
extern void Creat_file_structure(int T,char *Initial,FILE *yfile,FILE *afile,FILE *tfile,FILE *ofile,FILE
*wfile,FILE *hfile,FILE *tmfile);
extern void Runge_Kutta_Fehlberg( Base_Pair BP[],Base_Pair y2[],Base_Pair y1[],Base_Pair y0[],double
h,double t, Base Strand[],double X,double V,int T,double G,double Total_Energy[],int Transport_Stuy,int Job_ID);
extern void fft(int , double (*x)[2], double (*X)[2]);
extern void ifft(int, double (*x)[2], double (*X)[2]);
extern void Auto_correaltion(Base_Pair BP_MAIN[]);
extern Norm Norm_Calculation(Base_Pair BP[],Cell cell_donor);
extern void Final_Times_Print_matrix(Storage_Base_Pair Master_Output_Buffer[],FILE *yfile,FILE *tfile,FILE
*wfile,int Buffer_Size,int Buffer_Index);
```

(viii) Random Number Generators:

```
/* ------------------------------------------------------------------
 * This is an ANSI C library for multi-stream random number generation.
 * The use of this library is recommended as a replacement for the ANSI C
 * rand() and srand() functions, particularly in simulation applications
 * where the statistical 'goodness' of the random number generator is
 * important.  The library supplies 256 streams of random numbers; use
 * SelectStream(s) to switch between streams indexed s = 0,1,...,255.
 *
```

```
 * The streams must be initialized.  The recommended way to do this is by
 * using the function PlantSeeds(x) with the value of x used to initialize
 * the default stream and all other streams initialized automatically with
 * values dependent on the value of x.  The following convention is used
 * to initialize the default stream:
 *    if x > 0 then x is the state
 *    if x < 0 then the state is obtained from the system clock
 *    if x = 0 then the state is to be supplied interactively.
 *
 * The generator used in this library is a so-called 'Lehmer random number
 * generator' which returns a pseudo-random number uniformly distributed
 * 0.0 and 1.0.  The period is (m - 1) where m = 2,147,483,647 and the
 * smallest and largest possible values are (1 / m) and 1 - (1 / m)
 * respectively.  For more details see:
 *
 *       "Random Number Generators: Good Ones Are Hard To Find"
 *                  Steve Park and Keith Miller
 *               Communications of the ACM, October 1988
 *
 * Name           : rngs.c  (Random Number Generation - Multiple Streams)
 * Authors        : Steve Park & Dave Geyer
 * Language       : ANSI C
 * Latest Revision : 09-22-98
 * ----------------------------------------------------------------------
 */

#include <stdio.h>
#include <time.h>
#include "rngs.h"

#define MODULUS    2147483647 /* DON'T CHANGE THIS VALUE                */
#define MULTIPLIER 48271      /* DON'T CHANGE THIS VALUE                */
#define CHECK      399268537  /* DON'T CHANGE THIS VALUE                */
#define STREAMS    256        /* # of streams, DON'T CHANGE THIS VALUE  */
#define A256       22925      /* jump multiplier, DON'T CHANGE THIS VALUE */
#define DEFAULT    123456789  /* initial seed, use 0 < DEFAULT < MODULUS  */

static long seed[STREAMS] = {DEFAULT};  /* current state of each stream   */
static int  stream        = 0;          /* stream index, 0 is the default */
static int  initialized   = 0;          /* test for stream initialization */


  double Random(void)
/* ----------------------------------------------------------------
 * Random returns a pseudo-random real number uniformly distributed
 * between 0.0 and 1.0.
 * ----------------------------------------------------------------
 */
{
  const long Q = MODULUS / MULTIPLIER;
  const long R = MODULUS % MULTIPLIER;
        long t;

  t = MULTIPLIER * (seed[stream] % Q) - R * (seed[stream] / Q);
  if (t > 0)
    seed[stream] = t;
```

```
  else
    seed[stream] = t + MODULUS;
  return ((double) seed[stream] / MODULUS);
}


   void PlantSeeds(long x)
/* ------------------------------------------------------------------
 * Use this function to set the state of all the random number generator
 * streams by "planting" a sequence of states (seeds), one per stream,
 * with all states dictated by the state of the default stream.
 * The sequence of planted states is separated one from the next by
 * 8,367,782 calls to Random().
 * ------------------------------------------------------------------
 */
{
  const long Q = MODULUS / A256;
  const long R = MODULUS % A256;
        int  j;
        int  s;

  initialized = 1;
  s = stream;                           /* remember the current stream */
  SelectStream(0);                      /* change to stream 0          */
  PutSeed(x);                           /* set seed[0]                 */
  stream = s;                           /* reset the current stream    */
  for (j = 1; j < STREAMS; j++) {
    x = A256 * (seed[j - 1] % Q) - R * (seed[j - 1] / Q);
    if (x > 0)
      seed[j] = x;
    else
      seed[j] = x + MODULUS;
    }
}


   void PutSeed(long x)
/* ------------------------------------------------------------------
 * Use this function to set the state of the current random number
 * generator stream according to the following conventions:
 *    if x > 0 then x is the state (unless too large)
 *    if x < 0 then the state is obtained from the system clock
 *    if x = 0 then the state is to be supplied interactively
 * ------------------------------------------------------------------
 */
{
  char ok = 0;

  if (x > 0)
    x = x % MODULUS;                    /* correct if x is too large  */
  if (x < 0)
    x = ((unsigned long) time((time_t *) NULL)) % MODULUS;
  if (x == 0)
    while (!ok) {
      printf("\nEnter a positive integer seed (9 digits or less) >> ");
      scanf("%ld", &x);
```

```
    ok = (0 < x) && (x < MODULUS);
    if (!ok)
      printf("\nInput out of range ... try again\n");
   }
  seed[stream] = x;
}


  void GetSeed(long *x)
/* ---------------------------------------------------------------
 * Use this function to get the state of the current random number
 * generator stream.
 * ---------------------------------------------------------------
 */
{
  *x = seed[stream];
}


  void SelectStream(int index)
/* ---------------------------------------------------------------
 * Use this function to set the current random number generator
 * stream -- that stream from which the next random number will come.
 * ---------------------------------------------------------------
 */
{
  stream = ((unsigned int) index) % STREAMS;
  if ((initialized == 0) && (stream != 0))   /* protect against        */
    PlantSeeds(DEFAULT);                     /* un-initialized streams */
}


  void TestRandom(void)
/* ---------------------------------------------------------------
 * Use this (optional) function to test for a correct implementation.
 * ---------------------------------------------------------------
 */
{
  long   i;
  long   x;
  double u;
  char   ok = 0;

  SelectStream(0);                /* select the default stream */
  PutSeed(1);                    /* and set the state to 1    */
  for(i = 0; i < 10000; i++)
    u = Random();
  GetSeed(&x);                    /* get the new state value   */
  ok = (x == CHECK);              /* and check for correctness */

  SelectStream(1);                /* select stream 1               */
  PlantSeeds(1);                  /* set the state of all streams  */
  GetSeed(&x);                    /* get the state of stream 1     */
  ok = ok && (x == A256);         /* x should be the jump multiplier */
  if (ok)
    printf("\n The implementation of rngs.c is correct.\n\n");
```

```
  else
    printf("\n\a ERROR -- the implementation of rngs.c is not correct.\n\n");
}
```

(ix) ANSI C library for generating random variates from six discrete distributions

```
/* ------------------------------------------------------------------------
 * This is an ANSI C library for generating random variates from six discrete
 * distributions
 *
 *     Generator        Range (x)     Mean         Variance
 *
 *     Bernoulli(p)      x = 0,1       p           p*(1-p)
 *     Binomial(n, p)    x = 0,...,n   n*p         n*p*(1-p)
 *     Equilikely(a, b)  x = a,...,b   (a+b)/2      ((b-a+1)*(b-a+1)-1)/12
 *     Geometric(p)      x = 0,...     p/(1-p)      p/((1-p)*(1-p))
 *     Pascal(n, p)      x = 0,...     n*p/(1-p)    n*p/((1-p)*(1-p))
 *     Poisson(m)        x = 0,...     m            m
 *
 * and seven continuous distributions
 *
 *     Uniform(a, b)     a < x < b     (a + b)/2    (b - a)*(b - a)/12
 *     Exponential(m)    x > 0         m            m*m
 *     Erlang(n, b)      x > 0         n*b          n*b*b
 *     Normal(m, s)      all x         m            s*s
 *     Lognormal(a, b)   x > 0           see below
 *     Chisquare(n)      x > 0         n            2*n
 *     Student(n)        all x         0  (n > 1)   n/(n - 2)   (n > 2)
 *
 * For the a Lognormal(a, b) random variable, the mean and variance are
 *
 *                     mean = exp(a + 0.5*b*b)
 *                 variance = (exp(b*b) - 1) * exp(2*a + b*b)
 *
 * Name             : rvgs.c  (Random Variate GeneratorS)
 * Author           : Steve Park & Dave Geyer
 * Language         : ANSI C
 * Latest Revision  : 10-28-98
 * ------------------------------------------------------------------------
 */

#include <math.h>
#include "rngs.h"
#include "rvgs.h"


  long Bernoulli(double p)
/* ========================================================
 * Returns 1 with probability p or 0 with probability 1 - p.
 * NOTE: use 0.0 < p < 1.0
 * ========================================================
 */
{
  return ((Random() < (1.0 - p)) ? 0 : 1);
}
```

```
  long Binomial(long n, double p)
/* ============================================================
 * Returns a binomial distributed integer between 0 and n inclusive.
 * NOTE: use n > 0 and 0.0 < p < 1.0
 * ============================================================
 */
{
  long i, x = 0;

  for (i = 0; i < n; i++)
    x += Bernoulli(p);
  return (x);
}

  long Equilikely(long a, long b)
/* ============================================================================
 * Returns an equilikely distributed integer between a and b inclusive.
 * NOTE: use a < b
 * ============================================================================
 */
{
  return (a + (long) ((b - a + 1) * Random()));
}

  long Geometric(double p)
/* ========================================================
 * Returns a geometric distributed non-negative integer.
 * NOTE: use 0.0 < p < 1.0
 * ========================================================
 */
{
  return ((long) (log(1.0 - Random()) / log(p)));
}

  long Pascal(long n, double p)
/* ====================================================
 * Returns a Pascal distributed non-negative integer.
 * NOTE: use n > 0 and 0.0 < p < 1.0
 * ====================================================
 */
{
  long i, x = 0;

  for (i = 0; i < n; i++)
    x += Geometric(p);
  return (x);
}

  long Poisson(double m)
/* ====================================================
 * Returns a Poisson distributed non-negative integer.
 * NOTE: use m > 0
 * ====================================================
 */
{
```

```
  double t = 0.0;
  long   x = 0;

  while (t < m) {
    t += Exponential(1.0);
    x++;
  }
  return (x - 1);
}

   double Uniform(double a, double b)
/* ============================================================
 * Returns a uniformly distributed real number between a and b.
 * NOTE: use a < b
 * ============================================================
 */
{
  return (a + (b - a) * Random());
}

   double Exponential(double m)
/* ============================================================
 * Returns an exponentially distributed positive real number.
 * NOTE: use m > 0.0
 * ============================================================
 */
{
  return (-m * log(1.0 - Random()));
}

   double Erlang(long n, double b)
/* ============================================================
 * Returns an Erlang distributed positive real number.
 * NOTE: use n > 0 and b > 0.0
 * ============================================================
 */
{
  long   i;
  double x = 0.0;

  for (i = 0; i < n; i++)
    x += Exponential(b);
  return (x);
}

   double Normal(double m, double s)
/*
========================================================================
===
 * Returns a normal (Gaussian) distributed real number.
 * NOTE: use s > 0.0
 *
 * Uses a very accurate approximation of the normal idf due to Odeh & Evans,
 * J. Applied Statistics, 1974, vol 23, pp 96-97.
```

```
 *
=============================================================================
===
 */
{
  const double p0 = 0.322232431088;     const double q0 = 0.099348462606;
  const double p1 = 1.0;                const double q1 = 0.588581570495;
  const double p2 = 0.342242088547;     const double q2 = 0.531103462366;
  const double p3 = 0.204231210245e-1;  const double q3 = 0.103537752850;
  const double p4 = 0.453642210148e-4;  const double q4 = 0.385607006340e-2;
  double u, t, p, q, z;

  u   = Random();
  if (u < 0.5)
    t = sqrt(-2.0 * log(u));
  else
    t = sqrt(-2.0 * log(1.0 - u));
  p   = p0 + t * (p1 + t * (p2 + t * (p3 + t * p4)));
  q   = q0 + t * (q1 + t * (q2 + t * (q3 + t * q4)));
  if (u < 0.5)
    z = (p / q) - t;
  else
    z = t - (p / q);
  return (m + s * z);
}

   double Lognormal(double a, double b)
/* ========================================================
 * Returns a lognormal distributed positive real number.
 * NOTE: use b > 0.0
 * ========================================================
 */
{
  return (exp(a + b * Normal(0.0, 1.0)));
}

   double Chisquare(long n)
/* ========================================================
 * Returns a chi-square distributed positive real number.
 * NOTE: use n > 0
 * ========================================================
 */
{
  long   i;
  double z, x = 0.0;

  for (i = 0; i < n; i++) {
    z  = Normal(0.0, 1.0);
    x += z * z;
  }
  return (x);
}

   double Student(long n)
/* ============================================
 * Returns a student-t distributed real number.
```

```
 * NOTE: use n > 0
 * ==========================================
 */
{
  return (Normal(0.0, 1.0) / sqrt(Chisquare(n) / n));
}
```

(x) Compute the transfer integral

```
/*****************************************************************************/
/*
  Purpose:
    Compute the transfer integral
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
    07 July 2010
  Author:
          Neranjan Edirisinghe
*/
/*****************************************************************************/


# include "stochastic_rk.h"
double V_cal(double theta1, double r1,double theta2, double r2, int i,int Transport_Study)
{

          double nppp,nppq,Vppp,Vppq;
          double a,b,c,d,Rc;
          double ALF1,ALF2,BTA1,BTA2,GMA1,GMA2;
          double THE,R,L,V,ALPHA,R1,R2;
THE=36;
R=10;
L=3.34;
nppp=-2.26;
nppq=5.27;
Rc=0.97;
ALPHA=4.45;

R1=(r1/ALPHA+R);
R2=(r2/ALPHA+R);
ALF1=R1*sin(atan((L)/R1))*cos(theta1/(ALPHA*R)+THE);
BTA1=R1*sin(atan((L)/R1))*sin(theta1/(ALPHA*R)+THE);
GMA1=R1*cos(atan((L)/R1));
ALF2=R2*sin(atan(((2)*L)/R2))*cos(theta2/(ALPHA*R)+(2)*THE);
BTA2=R2*sin(atan(((2)*L)/R2))*sin(theta2/(ALPHA*R)+(2)*THE);
GMA2=R2*cos(atan(((2)*L)/R2));
a=(ALF1-ALF2);
b=(BTA1-BTA2);
c=(GMA1-GMA2);
d=(sqrt(pow(a,2)+pow(b,2)+pow(c,2)));
Vppp=nppp*7.62*exp(-d/Rc);
Vppq=nppq*7.62*exp(-d/Rc);
V=7.62*exp(-d/Rc)*((fabs(nppp)+nppq)*(pow(L,2)/(pow(d,2))-fabs(nppp)));
```

```
return V;
}
```

(xi) Compute the Norm and statistic

```
/**************************************************************************/
/*
  Purpose:
    Compute the Norm and statistic across all processors and gather information on to the one processor
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
    07 July 2010
  Author:
          Neranjan Edirisinghe
*/
/**************************************************************************/


# include "stochastic_rk.h"
Norm Norm_Calculation(Base_Pair BP[],Cell cell_donor)
{         Norm Data_Norm,Data_Norm_total;
          double Temp1,Temp2,Temp3,POW,XI,XR;
int j,size,myrank,M;


MPI_Comm_size ( MPI_COMM_WORLD, &size );
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
M=N/size;
Data_Norm.Participation_Function=0;
Data_Norm.Return_Probability=0;
Data_Norm.Shannon_Information_Entrophy=0;
Temp1=0;
Temp2=0;
Temp3=0;

          for (j=ghost;j<M+ghost;j++)
          {         XI=BP[j].XI;
                    XR=BP[j].XR;

                    POW=pow(XI,2)+pow(XR,2);

                    Temp1=Temp1+POW;
                    Temp2=Temp2-((POW)*log(POW));
                    Temp3=Temp3+pow(POW,2);
          }

          /*
          if (myrank==cell_donor.Sub_unit)
          {
                    XI=BP[cell_donor.Position+ghost-1].XI;
                    XR=BP[cell_donor.Position+ghost-1].XR;
                    Data_Norm.Return_Probability=pow(XI,2)+pow(XR,2);
```

```
	}
	*/
	Data_Norm.Return_Probability=Temp1;
	Data_Norm.Participation_Function=Temp3;
	Data_Norm.Shannon_Information_Entrophy=Temp2;
	MPI_Reduce(&Data_Norm, &Data_Norm_total,3,MPI_DOUBLE,MPI_SUM, 0, MPI_COMM_WORLD) ;
	MPI_Barrier(MPI_COMM_WORLD);
	return Data_Norm_total;
}
```

(xii) Gather data from all processors and print result into a file

```
/*****************************************************************************/
/*
  Purpose:
  Gather data from all processors and print result into a file
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
    07 July 2010
  Author:
	Neranjan Edirisinghe
*/
/*****************************************************************************/

#include "stochastic_rk.h"
void Final_Times_Print_matrix(Storage_Base_Pair Master_Output_Buffer[],FILE *yfile,FILE *tfile,FILE
*wfile,int Buffer_Size,int Buffer_Index)
{
int j,i,k,size,myrank,M,Globle_Index,Local_Index;
	div_t nh;
MPI_Comm_size ( MPI_COMM_WORLD, &size );
MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
M=N/size;
	for (j=0;j<Buffer_Size;j++)
	{
			for (i=0;i<size;i++)
			{
				for (k=0;k<M;k++)
				{
						nh = div(Buffer_Index+j, Buffer_Size);
						Local_Index=nh.rem;
						Globle_Index=Local_Index*M+i*Buffer_Size*M+k;



						fprintf(yfile,"%12.10f
",Master_Output_Buffer[Globle_Index].Displacement);
						fflush(yfile) ;
						fprintf(tfile,"%12.10f
",Master_Output_Buffer[Globle_Index].Twist_Angle);
						fflush(tfile) ;
						fprintf(wfile,"%12.10f
",Master_Output_Buffer[Globle_Index].Wave_Function);
						fflush(wfile) ;
```

```
                                        }
                                }
        fprintf(yfile," \n");
        fprintf(tfile," \n");
        fprintf(wfile," \n");
        }
```

(xiii) Print intermediate result

```
/*****************************************************************************/
/*
  Purpose:
  Print intermediat result into a file
  Licensing:
    This code is distributed under the GNU LGPL license.
  Modified:
    07 July 2010
  Author:
        Neranjan Edirisinghe
*/
/*****************************************************************************/

#include "stochastic_rk.h"
void Print_matrix(Base_Pair BP_MAIN[],FILE *yfile,FILE *afile,FILE *tfile,FILE *ofile,FILE *wfile,FILE *hfile)
{
int j;

        for (j=0;j<N;j++)
                        {


                                        fprintf(yfile,"%12.10f  ",BP_MAIN[j].Y);
                                        fflush(yfile) ;
                                        fprintf(afile,"%12.10f  ",BP_MAIN[j].A);
                                        fflush(afile) ;
                                        fprintf(tfile,"%12.10f  ",BP_MAIN[j].TH);
                                        fflush(tfile) ;
                                        fprintf(ofile,"%12.10f  ",BP_MAIN[j].OM);
                                        fflush(ofile) ;
                                        fprintf(wfile,"%12.10f
",pow(BP_MAIN[j].XR,2)+pow(BP_MAIN[j].XI,2));
                                        fflush(wfile) ;
                                        fprintf(hfile,"%12.10f
",pow(BP_MAIN[j].HR,2)+pow(BP_MAIN[j].HI,2));
                                        fflush(hfile) ;



                        }



        fprintf(yfile," \n");
        fflush(yfile) ;
        fprintf(afile," \n");
```

```
            fflush(afile) ;
            fprintf(tfile," \n");
            fflush(tfile) ;
            fprintf(ofile," \n");
            fflush(ofile) ;
            fprintf(wfile," \n");
            fflush(wfile) ;
            fprintf(hfile," \n");
            fflush(hfile) ;
}
```

(b) IV Characteristic of DNA

(i) Main Transport calculations (matlab)

```
function [CR,TRT,TST,ES,ER]=Main_Transport_calc(M,N,K)
%M=14
for STUDY_INPUT=N:K
SQ=ones(1,15);
DLT=zeros(1,15);
%Parameter_set
Index=0;
global Sigma_G;
global Sigma_C;
global Tg;
global Tc;
global STUDY;
Sigma_G=9.0;
Sigma_C=9.0;
Tg=3.5;
Tc=3.5;
STUDY=STUDY_INPUT;
ESym=sym('ESym');
if (STUDY==1)
   PSS=-.75:.1:.75
   else if (STUDY==2)
      PSS=-1.5:.25:1.5
      else if (STUDY==3)
      PSS=-1.5:.25:1.5
         else if (STUDY==4)
      PSS=-.75:.15:.75
            else if (STUDY==5)
      PSS=-.75:.15:.75
               else if (STUDY==6)
      PSS=-1.5:.25:1.5
                  else if (STUDY==7)
      PSS=-.75:.15:.75
                     else
                     PSS=1
                     end

                  end
               end
            end
         end
```

```
    end
end
```

```
for i=1:size(PSS,2);
   PS=PSS(1,i);
Index=Index+1;
   [eh,el,t1,t2,t3]=DNA_Structure_Build(M,SQ,DLT,PS,ESym)
```

(ii) Hamiltonian matrix formation

```
B=B_matrix_formation(M,eh,el,t1,t2,t3)
EV=subs(B,{ESym},8.9)
E0=eigs(EV,2*M)
E1=zeros(2*M,1);
TOL=sum(E1-E0);
TOL=10;
Tol=.1;
while TOL>Tol
   EV=subs(B,{ESym},abs(mean(E1(M:2*M,1))-mean(E0(M:2*M,1))));
   E1=eigs(EV,2*M);
   TOL=(abs(mean(E1(M:2*M,1))-mean(E0(M:2*M,1))))
   E0=E1;
end
```

```
BW=1;
[CR,TR,TS,ES,ER]=current_calculation(B,M,E0,PS,ESym) ;
  TRT(Index,:)=TR
  TST(Index,:)=TS;
  ERE(Index,:)=ER;
  ESE(Index,:)=ES;
  CRC(Index,:)=CR;
end
save(['TRT_' num2str(STUDY_INPUT) '.mat'],'TRT');
save(['TST_' num2str(STUDY_INPUT) '.mat'],'TST');
save(['ESE_' num2str(STUDY_INPUT) '.mat'],'ESE');
save(['ERE_' num2str(STUDY_INPUT) '.mat'],'ERE');
save(['CRC_' num2str(STUDY_INPUT) '.mat'],'CRC');
end
```

(iii) DNA Structure Build

```
function [eh,el,t1,t2,t3]=DNA_Structure_Build(M,SQ,DLT,PS,ESym)

[muuue,theta,C,tho,kb,ThL,ThR,eGCh,eGCl,eATh,eATl ...
   eGAh,eGAl,tGCh,tGChRs,tGChLs,tGChLa,tGChRa ...
   tGCl,tGClLs,tGClRs,tGClLa,tGClRa ...
   tGC,tGCLs,tGCRs,tGAs,tGAa ...
    GC,AT,GAs,GAa...
```

```
    ]=Parameter_set(M,PS,ESym);
VCT=V_T_CAL(theta,C);

for I=1:M

switch ( SQ(I) )

        case GC
        C=6;
            if (SQ(I+1)==GC)
               eh(1,I)=eGCh;
               el(1,I)=eGCl;
               t1(1,I)=tGCh;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);%;
              t2(1,I)=-tGC;%;
               t3(1,I)=tGCl;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);%;;
            elseif (SQ(I+1)==AT)
               eh(1,I)=eGCh;
               el(1,I)=eGCl;
               t1(1,I)=-0.218;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
               t2(1,I)=-0.001;
               t3(1,I)=0.066;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);

            elseif (SQ(I+1)==GAs)

               eh(1,I)=eGCh;
               el(1,I)=eGCl;
               t1(1,I)=0.254;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
               t2(1,I)=0.024;
               t3(1,I)=-0.153;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);


            else (SQ(I+1)==GAa);

               eh(1,I)=eGCh;
               el(1,I)=eGCl;
               t1(1,I)=-0.213;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
               t2(1,I)=0.024;
               t3(1,I)=-0.072;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
            end



        case AT
        C=6;
          if (SQ(I+1)==GC)

               eh(1,I)=eATh;
               el(1,I)=eATl;
               t1(1,I)=-0.102;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
              t2(1,I)=-0.063;
               t3(1,I)=0.067;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
            elseif (SQ(I+1)==AT)
               eh(1,I)=eATh;
               el(1,I)=eATl;
               t1(1,I)=-0.011;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
               t2(1,I)=0.054;
```

```
    t3(1,I)=-0.075;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);

  elseif (SQ(I+1)==GAa)

    eh(1,I)=eATh;
    el(1,I)=eATl;
    t1(1,I)=-.002;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
    t2(1,I)=0.054;
    t3(1,I)=-0.079;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);


  else (SQ(I+1)==GAs);

    eh(1,I)=eATh;
    el(1,I)=eATl;
    t1(1,I)=0.027;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
    t2(1,I)=-0.054;
    t3(1,I)=-0.092;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
  end



case GAs
 C=6;
   if (SQ(I+1)==GC)

    eh(1,I)=eGAh;
    el(1,I)=eGAl;
    t1(1,I)=-0.136;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
     %t1=VCT;
    t2(1,I)=0.091;
    t3(1,I)=-0.073;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
  elseif (SQ(I+1)==AT)
    eh(1,I)=eGAh;
    el(1,I)=eGAl;
    t1(1,I)=0.207;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
    t2(1,I)=-0.161;
    t3(1,I)=0.011;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);

  elseif (SQ(I+1)==GAs)

    eh(1,I)=eGAh;
    el(1,I)=eGAl;
    t1(1,I)=-.002;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
    t2(1,I)=0.054;
    t3(1,I)=-0.079;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);


  else (SQ(I+1)==GAa);

    eh(1,I)=eGAh;
    el(1,I)=eGAl;
    t1(1,I)=tGChLa;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
    t2(1,I)=tGAa;
    t3(1,I)=tGClLa;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
  end
```

```
        case GAa
         C=6;
           if (SQ(I+1)==GC)

               eh(1,I)=eGAh;
               el(1,I)=eGAl;
               t1(1,I)=0.146;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
               t2(1,I)=0.020;
               t3(1,I)=-0.071;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
           elseif (SQ(I+1)==AT)
               eh(1,I)=eGAh;
               el(1,I)=eGAl;
               t1(1,I)=-0.155;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
               t2(1,I)=0.131;
               t3(1,I)=-0.117;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
           elseif (SQ(I+1)==GAs)

               eh(1,I)=eGAh;
               el(1,I)=eGAl;
               t1(1,I)=tGChLs;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
               t2(1,I)=tGAs;
               t3(1,I)=tGClLs;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);


           else (SQ(I+1)==GAa);

               eh(1,I)=eGAh;
               el(1,I)=eGAl;
               t1(1,I)=tGChLa;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
               t2(1,I)=tGAa;
               t3(1,I)=tGClLa;%+(V_T_CAL(theta+(DLT(I)-DLT(I+1)),C)-VCT);
           end


end
end
```

(iv) current calculation

```
function [CR1,TR,TS,ER,ES]=current_calculation(B,M,E0,PS,ESym)
KKK=1;
STP=0;
V=0:.001:2;
   V1=0:.1:1;
   V2=0:.001:3;
   muL=-10.5;
   Bvol_Tem1=-10.5+V;
   Bvol_Tem2=-8.5+V1;
   Bvol_Tem3=-7+V2;
  muR=[Bvol_Tem1 Bvol_Tem2 Bvol_Tem3];
  FACTOR=2;
[muuue,theta,C,tho,kb,ThL,ThR,eGCh,eGCl,eATh,eATl ...
```

```
    eGAh,eGAl,tGCh,tGChRs,tGChLs,tGChLa,tGChRa ...
    tGCl,tGClLs,tGClRs,tGClLa,tGClRa ...
    tGC,tGCLs,tGCRs,tGAs,tGAa ...
     GC,AT,GAs,GAa...
     ]=Parameter_set(M,PS,ESym);
BW=1;



TCR1=zeros(1,size(muR,2));
CR1=zeros(1,size(muR,2));
  STEP_NO=M*100;
io=0;
DOS=0;
TRS=0;
STEP=abs(E0(1)-E0(M))/STEP_NO;
CONST1=STEP*4e-5/1e-9;
CONT=0;
OP=1;
for E=E0(1):.1:E0(M);
    for k=1:2
       A=zeros(2*M,2*M);
       A=E*eye(size(A,1))-subs(B,'ESym',E);
       eig(A)
       A=A+eye(size(A,1))*(-1)^k*sqrt(-1)*tho/BW;
          G(:,:,k)=inv(A);
    end
   CONT=CONT+1;
   T=300;
    Ga=squeeze(G(:,:,1));
    Gr=squeeze(G(:,:,2));
    TMP=trace(Gr*ThL*Ga*ThR);
    TRS=TRS+TMP;
    DOS=DOS-imag(Ga(KKK,KKK))/pi;
l=0;
TR(1,CONT)=abs(TMP);
ER(1,CONT)=E;

    fL=1./(exp((E-muL)/(kb*T))+1);
    fR=1./(exp((E-muR)/(kb*T))+1);
    l=l+1;
    TM2=CONST1*(fR-fL)*TMP;
    CR1=CR1+TM2;
    TCR1=TCR1+TM2;
    STP=STP+1;
    OP=OP+1
end
%subplot(2,1,1)
%hold on
%plot(E0(1):STEP:E0(M),TR)
%pause(0.1)
STP=0;
STEP=abs(E0(M+1)-E0(2*M))/STEP_NO;
CONST1=STEP*4e-5/1e-9;
CONT=0;
for E=E0(M+1):STEP:E0(2*M);
```

```
    for k=1:2
      A=zeros(2*M,2*M);
      A=E*eye(size(A,1))-subs(B,'ESym',E);
      A=A+eye(size(A,1))*(-1)^k*sqrt(-1)*tho/BW;
      G(:,:,k)=inv(A);
    end

   T=300;

    Ga=squeeze(G(:,:,1));
    Gr=squeeze(G(:,:,2));
    TMP=trace(Gr*ThL*Ga*ThR);
    TRS=TRS+TMP;
    DOS=DOS-imag(Ga(KKK,KKK))/pi;
l=0;
CONT=CONT+1;
TS(1,CONT)=abs(TMP);
ES(1,CONT)=E;

    fL=1./(exp((E-muL)/(kb*T))+1);
    fR=1./(exp((E-muR)/(kb*T))+1);
    l=l+1;
    TM2=CONST1*(fR-fL)*TMP;
    CR1=CR1+TM2;
    TCR1=TCR1+TM2;
    STP=STP+1;
end
%subplot(2,1,2)
%hold on
%plot(E0(M+1):STEP:E0(2*M),TS)
%pause(0.1)

end
```

(v) Parameter set

```
function [muuue,theta,C,tho,kb,ThL,ThR,eGCh,eGCl,eATh,eATl ...
    eGAh,eGAl,tGCh,tGChRs,tGChLs,tGChLa,tGChRa ...
    tGCl,tGClLs,tGClRs,tGClLa,tGClRa ...
    tGC,tGCLs,tGCRs,tGAs,tGAa ...
     GC,AT,GAs,GAa...
    ]=Parameter_set(M,PS,ESym)
global Sigma_G;
global Sigma_C;
global Tg;
global Tc;
global STUDY;
muuue=0;
theta=32;
C=6;
tho=10*1e-3;
h=4.13566733e-15;
kb=8.61e-5;
Tho=0;
    ThL=zeros(2*M,2*M);
```

```
    ThR=zeros(2*M,2*M);
    ThL(1,1)=tho;
    ThL(1,M+1)=tho;
    ThL(M+1,1)=tho;
    ThL(M+1,M+1)=tho;
    ThR(M,M)=tho;
    ThR(2*M,2*M)=tho;
    ThR(2*M,M)=tho;
    ThR(M,2*M)=tho;

if (STUDY==1)
 tGCh= -0.133+PS*(-0.133);
  tGCl= -0.041+PS*(-0.041);
else
 tGCh= -0.133;
  tGCl= -0.041;
end
 tGChLs= -0.213;
 tGChRs= 0.146;
 tGChLa= 0.254;
 tGChRa= -0.136;


 tGClLs= -0.072;
 tGClRs= -0.071;
 tGClLa= -0.153;
 tGClRa= -0.073;

 tGC= -0.05;
 tGCLs= 0.024;
 tGCRs= -0.025;
 tGAs =0.02;
 tGAa =0.091;
TG=Tg+9.4*(ESym-Sigma_G+i*Tho)/Tg;
 eGCh=( tGC+9.4+TG^2/(ESym-Sigma_G+i*Tho));
 TG=Tc+5.8*(ESym-Sigma_C+i*Tho)/Tc;
 eGCl=( tGC+5.8+TG^2/(ESym-Sigma_C+i*Tho));


 if (STUDY==2)
  TG=Tg+9.4*(ESym-(Sigma_G+PS)+i*Tho)/Tg;
  eGCh= ( tGC+9.4+TG^2/(ESym-(Sigma_G+PS)+i*Tho));
end
if (STUDY==3)
 TG=Tc+5.8*(ESym-(Sigma_C+PS)+i*Tho)/Tc;
 eGCl=( tGC+5.8+TG^2/(ESym-(Sigma_C+PS)+i*Tho));
end

if (STUDY==4)
 TG=Tg+PS+9.4*(ESym-Sigma_G+i*Tho)/(Tg+PS);
 eGCh=( tGC+9.4+TG^2/(ESym-Sigma_G+i*Tho));
end
if (STUDY==5)
 TG=Tc+PS+5.8*(ESym-Sigma_C+i*Tho)/(Tc+PS);
 eGCl=( tGC+5.8+TG^2/(ESym-Sigma_C)+i*Tho);
end
```

```
 if (STUDY==6)
  TG=Tg+9.4*(ESym-(Sigma_G+PS)+i*Tho)/Tg;
  eGCh= ( tGC+9.4+TG^2/(ESym-(Sigma_G+PS)+i*Tho));
  TG=Tc+5.8*(ESym-(Sigma_C+PS)+i*Tho)/Tc;
 eGCl=( tGC+5.8+TG^2/(ESym-(Sigma_C+PS)+i*Tho));
 end

 if (STUDY==7)
 TG=Tg+PS+9.4*(ESym-Sigma_G+i*Tho)/(Tg+PS);
 eGCh=( tGC+9.4+TG^2/(ESym-Sigma_G+i*Tho));
 TG=Tc+PS+5.8*(ESym-Sigma_C+i*Tho)/(Tc+PS);
 eGCl=( tGC+5.8+TG^2/(ESym-Sigma_C+i*Tho));
end

 eATh= -9.76-muuue;
 eATl= -6.56-muuue;
 eGAh= -9.4-muuue;
 eGAl=-5.84-muuue;

 GC= 1;
 AT= 2;
 GAs= 3;
 GAa= 4;


function B=B_matrix_formation(M,eh,el,t1,t2,t3)
 %B=zeros(2*M,2*M);
for I=1:M

                if ((I==1)||(I==M))
                B(I,I)=eh(1,I);
                B(M+I,M+I)=el(1,I);
                else
                B(I,I)=eh(1,I);%+i*tho/100;
                B(M+I,M+I)=el(1,I);
                end


                if I<M
                B(I,I+1)=-t1(1,I);
                B(I+1,I)=-t1(1,I);
                B(M+I,M+I+1)=-t3(1,I);
                B(M+I+1,M+I)=-t3(1,I);
                B(I,M+I+1)=-t2(1,I);
                B(M+I+1,I)=-t2(1,I);
                end


       T1(1,I)=t1(1,I);
end
```

(vi) Transfer integral calculation

```
function VVALU=V_T_CAL(theta,C)

    a0 =0;
    a1 =0;
    b1 =0;
    a2 =0;
    b2 =0;
    a3 =0;
    b3 =0;
    a4 =0;
    b4 =0;
    w =0;
  if (C==1.0)
   %/*TT*/
    a0 =-0.2867;
    a1 =-0.4697;
    a2=0.0;
    a3=0.0;
    b1 =0.2372;
    a2 =-0.1644;
    b2 =0.03453;
    b3=0.0;
    b4=0.0;
    w =0.06417;
    FI=1;

  end
  if (C==2.0)
  %/*CC*/
    a0 =-0.3326;
    a1 =-0.3204;
    a2=0.0;
    a3=0.0;
    b1 =0.1865;
    a2 =-0.1409;
    b2 =0.02608;
    b3=0.0;
    b4=0.0;
    w = 0.05618;
    FI=1;

  end
  if (C==3.0)
  %/*AA*/
    a0 =-0.2379;
    a1 =-0.4194;
    a2=0.0;
    a3=0.0;
    b1 =-0.01127;
    a2 =-0.07489;
    b2 =0.02547;
    b3=0.0;
```

```
  b4=0.0;
  w =0.05626;
  FI=1;

end
if (C==4.0)
%/*GG*/
  a0 =-0.2073;
  a1 =-0.3951;
  a2=0.0;
  a3=0.0;
  b1 =0.1232;
  a2 =-0.09205;
  b2 =0.003048;
  b3=0.0;
  b4=0.0;
  w =0.05438;
  FI=1;

end
if (C==5.0)
% /*AT*/
  a0 =0.09894;
  a1 =0.08126;
  b1 =-0.1539;
  a2 =0.08326;
  b2 =0.08645;
  a3 =0.01602;
  b3 =0.009397;
  a4 =0.003014;
  b4 =-0.001768;
  w = 0.06283;
  FI=4;

end
if (C==6.0)
%/*GC*/
  a0 =0.0841;
  a1 =0.1588;
  b1 =0.1426;
  a2 =-0.1499;
  b2 =0.01449;
  a3 =-0.02541;
  b3 =0.07448;
  a4 =0.02631;
  b4 =-0.0379;
  w = 0.09746;
  FI=4;

end
if (C==7.0)
%/*AC*/
  a0 =13.06;
  a1 =-8.802;
  b1 =-19.64;
  a2 =-7.779;
```

```
  b2 =8.36;
  a3 =3.241;
  b3 =1.074;
  a4 =-0.08756;
  b4 =-0.5004;
  w =0.03142;
  FI=4;

end
if (C==8.0)
%/*GA*/
   a0 =3.534e+009;
  a1 =-5.52e+009;
  b1 =1.247e+009;
  a2 =2.561e+009;
  b2 =-1.219e+009;
  a3 =-6.395e+008;
  b3 =5.028e+008;
  a4 =6.447e+007;
  b4 =-7.933e+007;
  w =-0.006177;
  FI=4;

end
if (C==9.0)
%/*CT*/
  a0 =-5067;
  a1 =5267;
  b1 =6278;
  a2 =736;
  b2 =-4168;
  a3 =-1105;
  b3 =637.8;
  a4 =161.8;
  b4 =58.44;
  w =0.02353;
  FI=4;

end
if (C==10.0)
%/*GT*/
  a0 =0.1099;
  a1 =0.01836;
  b1 =-0.03161;
  a2 =-0.1138;
  b2 =0.1188;
  a3 =0.06883;
  b3 =0.04597;
  a4 =-0.05873;
  b4 =-0.03044;
  w =0.1215;
  FI=4;

End
```

### (vii) Parameter fitting for transfer integral

```
VVALU=V_fun(a0,a1,a2,a3,a4,b1,b2,b3,b4,w,theta,FI);

function temp=V_fun(a0,a1,a2,a3,a4,b1,b2,b3,b4,w,x,FI)

if (FI==1)

    temp=a0+a1*cos(x*w)+b1*sin(x*w)+a2*cos(2*x*w)+b2*sin(2*x*w);
elseif (FI==4)


temp=a0+a1*cos(x*w)+b1*sin(x*w)+a2*cos(2*x*w)+b2*sin(2*x*w)+a3*cos(3*x*w)+b3*sin(3*x*w)+a4*cos(4*
x*w)+b4*sin(4*x*w);

end
```