This is to certify that the

thesis entitled

A Continuous Monte Carlo Method for
Simultanious Growth and
Equilibration in Gelation
presented by

Rajasinghe Nimalakirthi

has been accepted towards fulfillment
of the requirements for

M.S. degree in CHE

Major professor

Date 3/10/95

A CONTINUOUS MONTE CARLO METHOD FOR SIMULTANIOUS GROWTH AND EQUILIBRATION IN GELATION

By

Rajasinghe Nimalakirthi

A THESIS

submitted to
Michigan State University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Department of Chemical Engineering

1995

ABSTRACT

A CONTINUOUS MONTE CARLO METHOD FOR SIMULTANIOUS GROWTH AND EQUILIBRATION IN GELATION

By

Rajasinghe Nimalakirthi

We have proposed a new realistic model for kinetic gelation studies, and its computational feasibility was successfully studied by constant NVT-ensemble Monte Carlo simulation. A continuous simulation method for polymer network formation in the presence of cross-linking agents was developed with simultaneous equilibration.

Important and ignored aspects of earlier models, such as molecular mobility and realistic potentials were incorporated into our proposed model. Monomer and polymer mobility were allowed and the effect of polymer mobility on simultaneous equilibration and growth process was carefully studied. It was important to avoid large density changess during the growth. Such density changes can occur due to reduced mobility of polymer units and the shift of intramolecular potential coordinate away from that of intermolecular potential. Growth and equilibration did not compete in our model. Chain cyclizations were allowed, and it was shown that the cage trapping was rare during the early part of the growth. During this early growth the percentage of trapped centers was very much smaller than that predicted by previous gelation simulations. Our model gave a better insight of an internal structure as this was determined by the simultaneous equilibration process, and was governed by the employed potentials.

ACKNOWLEDGMENTS

My sincere thanks go to Professor Alec Scranton for his encouragement and support throughout this study. I would also like to thank my committee members Professor C. Lira and Professor S. D. Mahanti and for their valuable suggestions.

A special thanks goes to my colleague Mr. Scott Coons, for his suggestions regarding numerical modeling.

TABLE OF CONTENTS

LIST OF FIGURES

Figure 8. Total number of cross-links against MC time for $C_i = 0.004$, $C_t = 0.2$ (lower curve). $C_i = 0.004$, $C_t = 0.3$ (upper curve). 80% reduction of polymer displacement (with further reduction by a factor of 0.995 between each MC cycle). [C_i = initiator concentration, C_t= tetra-functional unit concentration]

Figure 9. Total number of cross-links against MC time for $C_i = 0.004$, $C_t = 0.3$ (- curve). Number of chain cyclizations against MC time (- - curve). 80% reduction of polymer displacement (with further reduction by a factor of 0.995 between each MC cycle). [C_i = initiator concentration, C_t= tetra-functional unit concentration]

1. INTRODUCTION

It is well known and practically important that, when sufficiently frequent branching or cross linking of chains occurs during the course of a polymerization the enhanced capacity for further growth of the larger molecules (due to their larger number of unreacted end groups) may lead ultimately to the formation of a spanning network structures so huge that the material changes rather abruptly from a fluid to a gel. The statistical theory of such reactions, developed by Flory [1,3] has had conspicuous success in predicting the gel point for the several polycondensation to which it has been applied [4].

At the gel point, a three dimensional network appears and the average cluster size is divergent. This is a very peculiar state of matter that is neither liquid nor solid but exists at the boundary between these states. For example, the viscosity is infinite. This incipient gel consists of a distribution of self-similar clusters of all sizes, and includes an infinitely massive cluster that is fractal on all length scales. If one were to pick a monomer at random, there would be zero probability that it belonged to the infinite cluster. One might now ask what happens to this perfectly self-similar arrangement as the system moves beyond the gel point [5].

Just beyond the gel point the very largest cluster combines with the infinite cluster to form a three-dimensional net work that is no longer a mass fractal on all length scales. Embedded in this network is a sol of fractal clusters whose average size continues to

decrease as the reaction proceeds and the largest remaining cluster accrete on to the network. The probability of a monomer belonging to this cluster (called the gel fraction) thus grows beyond the gel point until all monomers have reacted. It seems remarkable that if we are able to stop the reaction and extract the sol from the gel, the sol would have the same fractal dimension and self-similar size distribution as beneath the gel point.

Many authors have generalized an early statistical description of Flory and Stockmayer by relaxing some of the assumptions of ideal random branching. These models consider tree-like polymer structure built from monomer units according to probabilistic rules for bond formation. The recursive nature of the polymer structure is exploited to obtain equations for statistical averages. Some of the statistical computational schemes reported in the literature include the recursive method [6,7], the fragment method [8,9] and the probability generating function method [10]. Many of the computational schemes are based on the same underlying model which incorporates the assumptions of ideal random cross-linking, including conversion independent kinetics, termination by chain transfer or disproportionation, equal reactivity and no cyclization.

The statistical method using probability generating functions allows expressions for structural averages to be obtained most efficiently and systematically. Although statistical methods provide an efficient way to obtain structural averages, application of this method depends on the assumption of relatively small concentrations of the multifunctional cross-linking agent.

Kinetic descriptions of free radical cross-linking polymerizations have been reported by Mikos and collaborators [11,12], where reaction kinetics is described in terms of the concentrations of the mono, di and pendent vinyl species, and moment method is

used to calculate averages for the linear primary chains. Tobita and Hamielec [13] used a pseudo kinetic rate constant method to simplify the kinetic equations. These authors illustrated that many reaction nonidealities could be included in their analysis, including unequal reactivity, conversion dependent kinetics, and cyclization in an approximate manner. Because free radical polymerizations are kinetically controlled, the structure of the polymer chains depends upon the conversion at which they were formed. Additionally, the history of the process is important. These features are more readily incorporated into the kinetic descriptions. However, the biggest disadvantage of kinetic descriptions relative to statistical description is the inability of obtaining structural information about the gel. Both methods are based upon the mean field approach, and therefore cannot accurately account for heterogeneity or topological constraints [14].

2. PERCOLATION AND KINETIC GELATION MODELS

2.1. Percolation Models

The review article by Stauffer and collaborators [15] compares classical approach (Flory-Stockmayer type theory) with the percolation theories. They discussed a solution of monomers with functionality f>3; from each monomer may emanate zero to f bonds to neighboring molecules and thus this monomer may participate in the formation of a large cluster which is called a macromolecule. Two monomers in the same cluster or macromolecule are thus connected directly or indirectly (through other monomers in the same cluster) by such bonds where as two monomers in two different macromolecules are not connected by such bonds. They denoted the number of monomers in one macromolecule by s and then called this macromolecule also an s cluster; an isolated monomer without bonds to its neighbors is thus designated as an 1-cluster with s=1. Under certain conditions, an "infinite" cluster can be formed, a network which extends from one end of the sample to the other. This infinitely large macromolecule is called a gel; a collection of finite cluster is called a sol. A gel usually coexist with a sol; the finite clusters are then trapped in the interior of the gel.

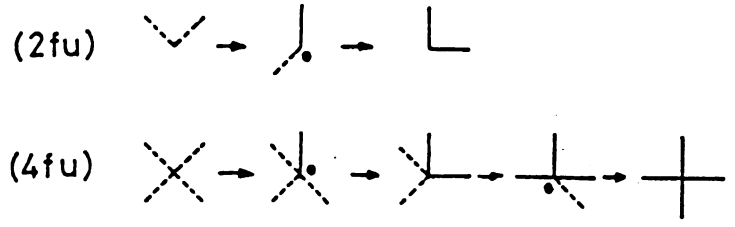
Gelation is the phase transition from a state without a gel to a state with a gel. The conversion factor p is the fraction of bonds which have been formed between the monomers of the system, i.e. the ratio of the actual number of bonds at the given moment to the maximally possible number of such bonds. Thus, for p=0, no bonds have been formed and all monomers remain isolated 1-clusters. In the other extreme p=1, all possible bonds between monomers have ben formed and thus all monomers in the system have clustered into one infinite network, with no sol phase left. Thus for small

p no gel is present whereas for p close to unity one such network exists. There is in general a sharp phase transition at some intermediate critical point $p=p_c$, where an infinite cluster starts to appear; a gel for p above p_c, a sol for p below p_c. This point $p=p_c$ is the gel point and may be the analog of a liquid-gas critical point: For p below p_c only a sol is present. But for p above p_c, sol and gel coexist with each other.

In percolation model, monomers are thought to occupy the sites of a periodic lattice, and between two nearest neighbors of lattice sites a bond is formed randomly with probability p.

2.2. Kinetic Gelation Models

In one of the earlier model, by Manneville and Seze [16] gelation process of bi-functional (2fu) and tetra-functional (4fu) units were studied in the absences of any solvent. In this model monomers are placed at the nodes of a simple cubic lattice.



Different states of 2fu and 4fu.

- dotted lines indicate functions available for linking
- heavy lines indicate functions used for linking with a neighboring unit
- dots indicate that the unit is an "active center" i.e. has a free radical able to open a double bond of a neighboring unit and to link with it (active centers were formed by introduction of free radicals in the system).

Before starting the simulation, the 4fu are distributed randomly in the sample with a concentration c_{4fu}, the other nodes being occupied by 2fu. The process is initiated by activation of a given concentration c_i of randomly chosen units.

At each step:

1) One of the active centers is selected at random
2) One of its eighteen first or second neighbors having available functions is selected at random
3) The two units are linked;
4) The active center is transferred to the neighbor or annihilated with the active center of the neighbor.

In this model existence of a "infinite" cluster is characterized by the formation of a gel phase.

An important drawback of many theories, in particular of both random percolation theory on a three-dimensional lattice and of the original Flory-Stockmayer theory, is their assumption that chemical bonds are formed randomly. In reality, for irreversible gelation the bonds are formed as a result of a kinetic process which contains both deterministic and random elements. The history of the process is important since the structure of the chains depends on the conversion at which they were formed.

A model for irreversible gelation was suggested by Herrmann, Stauffer and Landau [17]. In this model bifunctional units and tetra-functional units are put into an inert solvent. Bi-functional unit has one and tetra-functional has two outer carbon double bonds which can be opened. An initiator is included in the sol which dissociates into two radicals. Each radical can saturate by breaking up a carbon double bond but leaving

the other bond unsaturate. This free bond acts as new radical opening up another double bond, and thus in a series of reactions a chain is created. Chains can cross-link at the tetra-functional molecules, thus forming branched macromolecules. The gel point is reached when for the first time an infinite macromolecule appears forming the gel which has a finite shear modulus. If two unsaturated carbon atoms encounter each other they can form a bond, thus annihilating their radical character.

Each site of the simple cubic lattice is initially occupied randomly by one of three types of molecules: tetra-functional units with probability c_t, bifunctional units with probability c_b and zero-functional solvent molecules with probability c_s.

$$c_t+c_b+c_s=1$$

Chemical bonds can only be made between nearest neighbors on lattice. The functionality of each molecule or unit gives the maximum number of chemical bonds which it can form with its neighbors; thus the zero-functional solvent is chemically inert and the bifunctional units alone would only form chains without cross-links and without a transition to a gel. Gelation is made possible by the presence of tetra-functional units which allows for chain intersections and for the formation of larger networks of chemically bonded molecules.

The growth process is governed by the active center. Active centers are produced by the dissociation of more complex molecules. If this dissociation is relatively slow then new active centers may be formed during the gelation process; if it is relatively fast, all active centers are produced at the beginning of the reaction. In the special case of no solvent and no mobility of condensable units, all active centers in the beginning of the simulation are assumed to occur in pairs since they did not have

enough time to physically separate after their chemical dissociation. Thus initially a fraction c_i of the links connecting nearest neighbors in the lattice is assumed to be occupied randomly; and an occupied bond means that the two units connected by that link each carry an active center and are permanently bond together. Since the number of bonds is three times the number of sites in a simple cubic lattice, and since each initiator contributes two active centers, the total number of active centers is $6c_iL^3$ in a lattice with L^3 sites at an initiator concentration c_i. When the polymerizable units are mobile in a solution, active centers are highly mobile and distribution is random at the beginning, no longer pair wise, with the only resreiction that no unit carries more than one radical.

Gelation process is defined by the motion of active centers: each active center can move from its present site to a nearest neighbor site, and then the bond between its old site and the new site is called occupied. Groups of polymerizable units connected by occupied bonds are called clusters and correspond to macromolecules.

The motion of an active center to its nearest-neighbor site is random except that the new site must have at least one free functionality. That means at most three (one) of the four (two) possible bonds emanating from the tetra-functional neighbor are allowed to be occupied before the active center jumps to that neighbor. Solvent molecules never carry an active center and are never connected by occupied bonds. Should the active center jump to a site carrying another active center, the two radicals annihilate each other and the bond between the two sites become occupied. Thus the number of occupied bonds increases with time, and these bonds trace the motion of active centers, the number of which decreases in time due to annihilation. Active centers can also become trapped even if they are not annihilated, if all their neighbors are chemically saturated, that is, have

all their possible two or four bonds occupied.

Since the occupied bonds describe the path of motion of an active center they are spatially correlated, in contrast to random percolation.

Any finite mobility is allowed in the model with help of the solvent molecules. Each solvent site may exchange place with a neighbor, be it another solvent molecule or a polymerizable monomer. A movement of a polymerisable unit is allowed only if it is connected by at most one occupied bond to another unit.

In contrast to earlier studies of Manneville and de Seze, this model allows for bonds between nearest neighbors only, and the initial distribution of initiators has a constraint.

The method is similar to Monte Carlo studies of random percolation. At the beginning of each simulation they randomly distributed tetra-functional units with probability c_t on a simple cubic lattice of L× L×L sites, with L up to 60. All other sites are occupied by bifunctional units with a concentration c_b or by solvent molecules with a concentration c_s. The positions of bi- and tetra-functional units are fixed throughout the simulation, with the exception of some runs. In the case of c_t=1 all sites are occupied by tetra-functional units but even then the system does not corresponds to random percolation. One reason is that in random percolation on this simple cubic lattice each site has functionality six, whereas these sites are only tetra-functional. In this model bonds are formed by a process of gradual initiation of neighboring bonds in contrast to random percolation, where bonds are formed randomly.

Initially no unit can have more than one occupied bond, in other words, the initiators are not allowed to be nearest neighbors; apart from that restriction, the spatial

distribution of initiators is taken as random. The kinetic process following the initial distribution of radicals is defined as follows: each initially occupied bond is regarded to connect a pair of free radicals. At each time unit (monte carlo step per radical), a radical randomly selects one of the six bonds emanating from this radical lattice site. This newly selected bond leads to another neighboring lattice site. Now one of two possibilities occurs. In the first case this neighboring lattice site has all its two or four bonds occupied, for bi- and tetra-functional lattice sites, respectively. In this case no additional bond can be occupied for this neighbor, and the original radical stays at its old place without creating any new occupied bond. In the other case, the number of occupied bonds emanating from this neighbor is smaller than the functionality of their neighbor. Then the bond connecting the original radical and its selected neighbor is regarded as occupied. If in this case of a successful attempt to create a new bond the neighbor happened to be another radical, the two radicals are regarded as annihilating each other.

In one Monte Carlo attempt, a radical site was selected randomly. If all its six neighbors are already saturated, then the radical is called trapped. If all radicals in the system are trapped, then no change will occur even if we make an infinite number of attempts. Initially, beside isolated monomers we have only clusters of size two, that is macromolecules containing two lattice sites connected by the initially occupied bonds. During the growth process described above the clusters can grow by the addition of one previously isolated site, or by coalescence of two clusters. Thus during the simulation it is rather easy to keep track of the number n_s of clusters containing s monomers each. s is proportional to the molecular weight if the bifunctional and tetra-functional units have the same weight; otherwise s should be called the degree of polymerization for

that particular macromolecule.

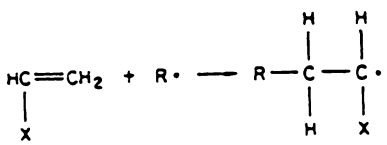
An improved model was suggested by Bansil, Herrmann and Stauffer [18] which doesn't assume that the bond formation is completely random, and it does incorporate the kinetic aspect of irreversible gelation. They considered the effects of incorporating solvent and mobility of monomers on the kinetics of free radical initiated copolymerization of a schematic vinyl and divinyl monomer.

The steps of their simulation was as follows.

1) Distribution of vinyl (or bifunctional) monomers, (b), divinyl (or tetra-functional) monomers (t), and solvent molecules (s), with concentration C_b, C_t and C_s respectively, on a simple cubic lattice with L^3 sites with the conservation condition

$$C_b+C_t+C_s=1$$

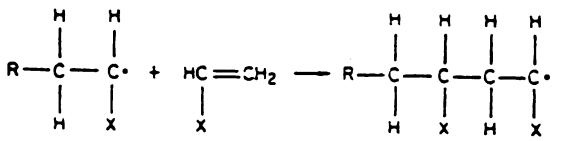
2) Growth process was initiated by randomly placing a free radical on a fraction C_i of of the bi- and tetra-functional monomers and drawing a bond along the lattice between the monomer and the radical to represent one of the reacted functionalities of the bi or tetra-functional monomer.



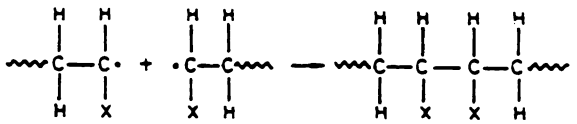
X denotes the side chain of the vinyl monomer and $R^{\bullet}$ the free radical obtained by the dissociation of the initiator.

3) The polymer growth process, the reaction is simulated by transferring a randomly chosen radical from its present position to a nearest-neighbor site along one of the six

bonds emanating from each lattice site and occupying this bond, provided the neighbor chosen is not occupied by solvent molecule and has at least one free functionality left. If these conditions are not met, the radical stays, for the present time, on the initial site.



4) If the nearest neighbor site also has a radical on it, then the two radicals combine, thereby simulating the reaction of chain termination by combination.



Molecular mobility was taken into account by allowing diffusion of monomers. They allowed each solvent molecule to exchange position with a nearest-neighbor molecule, provided the nearest-neighbor site is occupied by an unreacted bi- or tetra-functional monomer or by another solvent molecule. The growth process is continued until a specified number of bonds has been produced. Growth process terminates due to the trapping of radicals. This trapping process occurs since a radical attempting to transfer may find all its neighbor sites occupied either by solvent molecules or by fully reacted monomers. If all radicals in the system are trapped, then no further growth is possible, and the reaction is terminated. The entire process can be simulated

many times by starting with a new initial distribution of monomers on the lattice.

During free-radical cross-linking polymerization microgel particles are formed. One reason for this is the inherent inhomogeneity of the growth process. In kinetic gelation model, conformational rearrangements are not taken into account, so that the approach to a quasi-equilibrium state by diffusion of polymer segments is not described. Diffusion control of termination at low conversion in solvents has been demonstrated experimentally by the dependence of the termination constant on the solvent viscosity. In general, we may expect that the absence of conformational rearrangements by diffusion is an important drawback of the kinetic gelation model. In particular, diffusion could lead to a stronger compaction of microgel particles. Growth model without diffusion may be applicable as a model for the cyclization of a single chain. In the kinetic gelation model free-radical cross-linking polymerization of many chains is studied. Diffusion will increase the termination rate and decrease the kinetic chain length. Thus a model without diffusion is expected to overestimate the inhomogeneity. In fact it is well-known experimentally that the kinetic chain length increases when diffusion of radical units is slowed down at increasing conversion; at high conversion the kinetic chain length decreases due to the enhanced trapping probability and the restricted diffusion of the monomer. In the kinetic gelation model, which does not contain diffusion, only the latter effect is present [19].

In one of the recent models by Bowman and Peppas [20] a face-centered cubic lattice was chosen instead of the simple-cubic lattice used by past kinetic gelation simulations. This change allows for a site to have twelve nearest neighbors instead of only six. A second basic change is the incorporation of void sites in the simulation in

order to represent free volume. Both of these changes allow for more facile mobility of the different species.

Other changes which have been made are the inclusion of monomer and initiator molecules which occupy multiple sites. By having monomers occupy multiple sites, the multiple functional groups of the monomer molecule are separated by a finite amount as opposed to being represented at the same site. The model utilizes an initiator molecule which decays into two radicals instead of generating a single radical upon decay. Finally, the simulation includes mobility of all species including the reacted polymer.

Within the simulation a single time step is the period of time required to examine each active radical and one nearest-neighbor site to determine if reaction and propagation will occur. One of the key developments of Bowman and Peppas [20] model is the ability of all species in the reaction to move.

Monomer, initiator and polymer may move and diffuse with the only constraints being that all bonds must be preserved and movement must occur from an occupied site to an unoccupied site. Mobility increases as void sites increase and as the bonding between sites decreases.

The specific mechanism of mobility is as follows. First, two nearest-neighbor sites, site no. 1 and site no.2, are selected random. Site no. 1 is designated as the site from which movement will occur while site no. 2 is designated as the site which movement will occur. Movement will occur if all the necessary conditions are met.

1) Site no. 1 is occupied.

2) Site no. 2 is a void space.

3) all sites to which site no. 1 is bonded are nearest neighbors of site no. 2.

During each time step a fraction of sites will be selected to be site no. 1 and a nearest-neighbor site is selected as site no. 2. If movement is possible as described by the above conditions, then it will occur. As a part of movement, bonds are also changed to ensure that bonds are between sites. As mobility increases, the heterogeneity will decrease while the tendency to trap radicals will also decrease. This will lead to a higher maximum conversion being reached as the polymer becomes more mobile. Mobility can be increased either by increasing the fraction of sites examined for mobility checks during each time step.

After the radicals have been generated from initiator decay, they are able to propagate via reaction with any nearest-neighbor functional group. In turn, if a second radical is on a nearest-neighbor site the radicals may react with each other and terminate. During each time step every radical that remains active may propagate or terminate. If the adjacent site is a void, an unreactive monomer link, or a portion of the polymer, that radical will remain inactive until at least the next time step. If, however, the adjacent site is an unreacted functional group, propagation of the radical will occur from its present site to the adjacent site. The parameter that must be set for each simulation are the initiator mole fraction, the fraction of sites which are void, the lattice size, and the fraction of sites which are inspected during each time step for mobility.

Despite the improvements in monomer size, initiation mechanism, and species mobility it is necessary to understand that this simulation still contains some very unrealistic elements. The first and the most obvious short coming of this model is that it is performed on a lattice. Polymerization do not occur on a lattice but rather the

molecules involved take on a continuous distribution of orientations. Also, the model does not account for forces and potentials at the molecular level, and the size of the monomer is still not accurately represented. The addition of a molecular size into the model by incorporating a monomer that occupies a number of sites is a great advance, but it will only approach the behavior of the actual monomer molecules.

2.3. Monte Carlo Simulations

The goal of our simulation effort describe in following chapters is to account for molecular mobility in a realistic manner. Rather than fixing reactive monomer sites in space as in most kinetic gelation simulations reported in literature, in our simulations the monomer units will be allowed to move freely in three-dimensional space as they interact according to defined intermolecular potentials. A typical simulation will include several hundred monomer units interacting in a cell with periodic boundary conditions. Spherical as well as dumbbell shaped molecules will be allowed to interact according to Lennard-Jones potentials.

Our first attempt is to calculate the potential energy in a system of Lennard-Jones atoms. The Lennard-Jones potential is given by the formula.

$$v^{LJ}(r) = 4\varepsilon((\sigma/r)^{12} - (\sigma/r)^{6}) \qquad (1)$$

ε = Well depth

σ = Radius at which potential equal to zero

(The well depth is often quoted in units of temperature as ε/k_B, where k_B is Boltzmann's constant)

We store the coordinate vectors of our atoms in three FORTRAN arrays RX(I), RY(I) and RZ(I), with the particle index I varying from 1 to N. For the Lennard-Jones

potential it is useful to have precomputed the value of σ^2, which is stored in the variable SIGSQ. The potential energy will be stored in a variable V, which is zeroed initially, and is then accumulated in a double loop over all distinct pairs of atoms, taking care to count each pair only once. Information about code development is given in reference 21 and here we highlight some of the important steps given there.

Periodic Boundary Conditions:

A cubic box is replicated throughout space to form an infinite lattice. In the course of the simulation, as a molecule moves in the original box, its periodic image in each of the neighboring boxes moves in exactly the same way. Thus, as a molecule leaves the central box, one of its images will enter through the opposite face. There are no walls at the boundary of the central box, and no surface molecules. This box simply forms a convenient axis system for measuring the coordinates of the N molecules. The Figure 1 (Figure 1.9 in reference 21) illustrates this in two dimension. It is not necessary to store the coordinates of all the images in a simulation, just the molecules in the central box. When a molecule leaves the box by crossing a boundary, attention may be switched to the image just entering. For a fluid of Lennard-Jones atoms, it should be possible to perform a simulation in a cubic box of side $L=6\sigma$, without a particle being able to 'sense' the symmetry of the periodic lattice. We use a cubic box in our simulation because of its geometrical simplicity.

Potential Truncation:

The heart of the Monte Carlo program involves the calculation of the potential energy of a particular configuration. We must include interactions between molecule 1

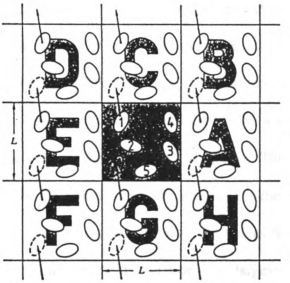


Figure 1. A two-dimensional periodic system. Molecules can enter and leave each box across each of the four edges (From Figure 1.9 in reference 21).

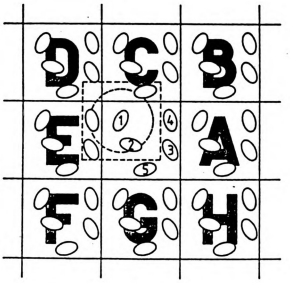


Figure 2. The minimum image convention in a two-dimensional system. The central box contains five molecules. The 'box' constructed with molecule 1 at its center also contains five molecules. The dashed circle represents a potential cutoff (From Figure 1.12 in reference 21).

and every other molecule i in the simulation box. In principle this is an (N-1) number of terms. For a short-range potential energy function, we restrict this summation by making an approximation. Consider molecule 1 in Figure 2 (Figure 1.12 in reference 21) to rest at center of a region which has the same size and shape as the basic simulation box. Molecule 1 interacts with all the molecules whose centers lie within this region, that is with the closest periodic images of the other N-1 molecules. This is called the 'minimum image convention'. In the minimum image convention, then the calculation of the potential energy due to pairwise-additive interactions involves N(N-1)/2 terms. A further approximation significantly improves this situation. The largest contribution to the potential comes from neighbors close to the molecule of interest, and for short range potential we apply a spherical cutoff. This means setting the pair potential to zero for $r>r_c$, where r_c is the cutoff distance. In a cubic simulation box of side L, the number of neighbors explicitly considered is reduced by a factor of approximately $4\pi r^3_c /3L^3$, and this may be a substantial saving. The cutoff distance must be no greater than L/2 for consistency with the minimum image convention.

Initially, the N molecules in the simulation lie within a cubic box of side L, with the origin at its center, all coordinates lie in the range (-L/2,L/2). As the simulation proceeds, these molecules move about the infinite periodic system. When a molecule leaves the box by crossing one of the boundaries, it is usual to switch attention to the image molecule entering the box, by simply adding L to, or subtracting L from, the approximate coordinate. For the coding information of this and the minimum image convention we refer the reader to reference 21.

We set the pair potential to zero if the pair separation lies outside some distance r_c.

It is easy to compare the square of the particle separation r_{ij} and, compare this with the square of r_c.

Metropolis Monte Carlo Method:

Classical statistics is assumed, only two-body forces are considered, and the potential field of a molecule is assumed spherically symmetric. System consists of a cube containing N particles. If we know the positions of the N particles in the cube, we can easily calculate, for example, the potential energy of the system (22)

$$V = (1/2) \sum_{\substack{i=1 \\ i \neq j}}^{N} \sum_{j=1}^{N} V(d_{ij}) \qquad (2)$$

(Here V is the potential between molecules, and d_{ij} is the minimum distance between particles i and j)

In order to calculate the properties of our system we use the canonical ensemble. So, to calculate the equilibrium value of any quantity of interest F,

$$\overline{F} = \int F \exp\left(-\frac{E}{kT}\right) d^{3N}p d^{3N}q \Big/ \int \exp\left(-\frac{E}{kT}\right) d^{3N}p d^{3N}q \qquad (3)$$

Where $d^{3N}pd^{3N}q$ is a volume element in the 6N-dimensional phase space. Moreover, since forces between particles are velocity-independent, the momentum integrals may be separated off, and we need perform only the integration over the 3N-dimensional configuration space. The Monte Carlo method for many-dimensional integrals consists simply of integrating over a random sampling of points instead of over a regular array of points.

Thus the most naive method of carrying out the integration would be to put each of the N particles at a random position in the cube, then calculate the energy of the system,

and give this configuration a weight exp(-E/kT). This method, however is not practical for close-packed configuration, since with high probability we choose a configuration where exp(-E/kT) is very small; hence a configuration of very low weight. So the method we employ is actually a modified Monte Carlo scheme, where instead of choosing configurations randomly, then weighting them with exp(-E/kT), we choose configurations with a probability exp(-E/kT) and weight them evenly.

We place the N particles in any configuration, for example, in a regular lattice. Then we move each of the particles in succession according to the following prescription:

$$\begin{aligned} X &\rightarrow X + \alpha\xi_1 \\ Y &\rightarrow Y + \alpha\xi_2 \\ Z &\rightarrow Z + \alpha\xi_3 \end{aligned} \qquad (4)$$

where α is the maximum allowed displacement, which for the sake of this argument is arbitrary, and ξ_1, ξ_2 and $\xi_3$are random numbers between (-1) and 1. Then, after we move a particle, it is equally likely to be any where within a cube of side 2α centered about its original position.

We then calculate the change in energy of the system E, which is caused by the move. If $E < 0$, if the move would bring the system to a state of lower energy, we allow the move and put the particle in its new position. If $E > 0$, we allow the move with probability exp(- E/kT); we take a random number ξ between 0 and 1, and if $\xi <$ exp(- E/kT), we move the particle to its new position. If $\xi >$ exp(- E/kT), we return it to its old position. Then, whether the move has been allowed or not, whether we are in a different configuration or in the original configuration, we consider that we are in a new configuration for the purpose of taking our averages. So

$$F = (1/M)\, \Sigma^{M}_{j=1}\, F_j \qquad (5)$$

where F_j is the value of the property F of the system after the jth move is carried out according to the complete prescription above. Having attempted to move a particle we proceed similarly with the next one.

A common practice in MC simulation is to select the atoms to move sequentially rather than randomly. This cuts down on the amount of random number generation and is an equally valid method of generating the correctly weighted states. The simulation of hard spheres is particularly easy using the MC method. The same Metropolis procedure is used, except that, in this case, the overlap of two spheres results in an infinite positive energy change and exp(- E/kT)=0. All trial moves involving an overlap are immediately rejected since exp(- E/kT) would be smaller than any random number generated on (0,1). Equally all moves that do not involve overlap are immediately accepted. As before in the case of a rejection the old configuration is recounted in the average (21).

Hard molecule fluids have played an important role in the development of our understanding of the liquid state. They have served as a useful check of statistical theories and have often provided the underlying fluid structure in perturbation theories for realistic molecular systems. The results of those theories have been successfully applied to many problems of simple liquids, to mixtures, and to diatomic liquids.

In the MC simulation of a molecular liquid the underlying matrix of the Markov chain is altered to allow moves which usually consists of a combined translation and rotation of one molecule. Chains involving a number of purely translational and purely rotational steps are perfectly proper but are not usually exploited in the simulation of molecular liquids. The translational part of the move is carried out by randomly

displacing the center of mass of a molecule along each of the space-fixed axes. As before the maximum displacement is governed by the adjustable parameter δr_{max}.

A method for changing the orientation of a molecule has been suggested by Jansoone [23]. It is more convenient to represent the molecular orientation by a unit vector **e** fixed in the molecule. The new trial orientation, **e** , is chosen randomly and uniformly on a region of the surface of a sphere with the constraint that,

$$1 - \mathbf{e}^n_i \cdot \mathbf{e}^m_i < d << 1. \qquad (6)$$

d controls the size of the maximum displacement, and a sensible first guess for this parameter is 0.2 [21]. There are a number of methods for generating a random vector on the surface of a sphere. The simplest of these uses the acceptance-rejection technique of Von-Newmann [24]. The procedure is iterative.

a) Generate three uniform random variates, ζ_1, ζ_2, and ζ_3, on (0,1);

b) Calculate $\zeta_i = 1-2\xi_i$ for i=1,3 so that the vector $\bar{\zeta}=(\zeta_1, \zeta_2, \zeta_3)$ is distributed uniformly in a cube of side 2, centered at the origin;

c) Form the sum $\zeta^2 = \zeta^2_1 + \zeta^2_2 + \zeta^2_3$

d) For $\zeta^2 < 1$ (inside the inscribed sphere) take $\bar{\zeta} =(\zeta_1 /\zeta, \zeta_2 /\zeta, \zeta_3 /\zeta)$ as the vector;

e) For $\zeta^2 > 1$ reject the vector and return to step (a).

Marsagilia [21] has suggested an interesting improvement;

a) Generate two uniform random variates ξ_1, ξ_1 on (0,1);

b) Calculate $\zeta_i =1-2\xi_i$ for i=1,2

c) For $\zeta^2 < 1$ take the vector, $\bar{\zeta} = [2\,\zeta_1(1-\zeta^2)^{1/2}, \zeta_2\,(1-\zeta^2)^{1/2}, 1-2\zeta^2]$;

d) For $\zeta^2 > 1$, reject and return to step (a).

A computer program developed to calculate Lennard-Jones potential of hard dumbbell system, by following Marsagilia's mathematical method is included in the appendix D.

(Note: This is a modification of the original program in reference 21)

One difficulty with MC methods for molecular fluids is that there are usually a number of parameters governing the maximum translational and orientational displacement of a molecule during a move. As usual these parameters can be adjusted automatically to give an acceptance rate of 0.5, but there is not a unique set of maximum displacement parameters which will achieve this. A sensible set of values is best obtained by trial and error for the particular simulation in hand. In our simulations maximum displacement was allowed to adjusted automatically to obtain 50% acceptance.

3. RECENT ADVANCES IN POLYMERIC GLASS STRUCTURES

Molecular dynamics simulation of bulk liquid and glass of long-chain molecules has been performed by Rigby and Roe [25] using a fairly realistic off-lattice chain model. Initial configurations of highly interwined chains were constructed by first generating a random distribution of points representing the centers of CH_2 groups (in the manner usually employed for mono-atomic systems). Groups in close proximity were joined to form several polymer chains. Individual chains are modeled as sequences of spherical segments connected by spring like valence bonds subject to the potential energy,

$$V_{b,j} = (1/2)K_b(l_j - l_o)^2, \tag{7}$$

where lo is the equilibrium bond length. The valence angle between successive pairs of bonds is maintained close to the tetrahedral value o by a potential quadratic in Cosθ:

$$V_{\theta,j}=(1/2)k\ (\text{Cos}\ \theta_j - \text{Cos}\ \theta_o)^2. \tag{8}$$

Finally, nonbonded interactions, between segments in different chains and between segments separated by more than three bonds along the chain backbone, are given according to a truncated Lennard-Jones potential. Simulation program was run with weakened force constants for all potentials (except for the LJ potential which was maintained "full strength" at all stages). Force constants were then gradually increased to their final values as the C-C bond lengths and the C-C-C valence angles decreased towards their equilibrium values.

An extension of Rigby and Roe method was recently carried out by Khare, Paulaitis and Lustig [26]. Their approach for generating glass structures was carried out in three stages. In the first stage, the monomers are initially placed in the simulation box in an order array. The system is then heated and relaxed by molecular

dynamics at an elevated temperature. The purpose at this stage of the procedure is to achieve a uniform spatial distribution and a random orientational distribution of monomers in the simulation box, and not an equilibrium liquid.

A polymerization of this configuration is then carried out in the second stage by connecting spatially adjacent monomers. Periodic boundary conditions are employed to eliminate surface effects, and the minimum image convention is used. The problem of connecting a given number of monomers to form the shortest polymer chain is similar to the classical "Traveling Salesman" problem. Traveling Salesman problem has been solved previously by simulated annealing, a multivariable optimization technique based on the Monte Carlo method. After connecting monomers in an initial polymerization sequence, simulated annealing is carried out by randomly altering this sequence and evaluating the cost of each move in terms of its effect on the total chain length of the polymer. Moves are accepted or rejected on the basis of standard Metropolis probabilities.

Many of the bond lengths and bond angles in the polymer chain backbone at the end of this stage will have values far from their equilibrium values, and hence the potential energy of this structure must be minimized primarily with respect to these degrees of freedom. Structures are relaxed using a combination of energy minimization and molecular dynamics at elevated temperature.

An non-lattice Monte Carlo method for growth and equilibration of dense liquids consisting of polymer chains with realistic intramolecular architecture was studied by Gupta, Westermann and Bitsanis [27]. The method succeeded in growing alkane chain length up to 25. They ignored the torsional potential until all the chains have grown to their full length. After selecting a chain randomly, a decision is made on

whether to further grow the chin or to attempt an equilibration move. At this stage growth and equilibration compete. Growth and equilibration moves are accepted and rejected unless they results in physical overlap between segments. Attempts to grow all the chains without simultaneous equilibration are bound to fail.

4. A NEW MODEL FOR GELATION

4.1. Description of the Model

We have developed a new model for free radical cross-linking polymerizations based upon Monte Carlo simulations. Bi-functional and tetra-functional units were considered as spheres. Monomers were initially placed in a simulation box in a cubic lattice. A random distribution and orientations were achieved by Monte Carlo simulation. Before the initiation, monomers interacted via Lennard-Jones potential. Predetermined number of monomers was randomly initiated to create active centers. The growth of linear polymer chains were achieved by bonding physically closest monomers to the active centers. These centers became active centers, and propagation occurred. The nearest neighbor status was examined before bond formation, so that the same monomer could not be bonded to more than one active center at a given instant of time (within a given Monte Carlo cycle). If at any given instant of time the same monomer was closest to more than one active center, the closest active center was bonded to that monomer and the next nearest neighbor(s) was (were) bonded to other(s). Cross-linking was made possible whenever an already bonded tetra-functional unit in one chain became the closest unit to the active center of another chain. The possibility for a given tetra-functional unit to form two consecutive bonds with the same monomer was avoided.

Within each Monte Carlo cycle all particle displacements were accepted or rejected according to their energies, and all active centers were allowed to form one bond each (provided bond length is less than reaction distance). Covalent bonds within individual chains were modeled by the harmonic oscillator potential with predetermined force constants. Along the chain backbone for non-neighboring units the

Lennard-Jones potential was not evaluated in our model because it usually falls beyond the cutoff region (r_c=2.2σ, and our monomer units consist of at least 2 c-c double bonds). For all other units, LJ potential was maintained.

In our model, mobility of both monomer and polymer were allowed in a realistic manner throughout the polymer growth. The maximum displacement allowed within the Monte Carlo simulation was reduced by a predetermined factor for all bonded units. This was done by multiplication of maximum displacement of bonded units by a factor of 0.8. In addition, bonded unit displacement was further reduced by a very small factor between each Monte Carlo cycle. This was done by multiplying the current factor by 0.995 within each MC cycle. This will account for the bulkiness of the polymer as it grows. Any displacement which moved monomers within the overlapping distance was immediately rejected. Bond formation was always carried out with the closest neighbor, provided it fell within a predetermined maximum bond length (reaction distance). As a result, the length of the growing linear chains could differ at any given instant of time, as well as at the end of the simulation. The maximum bond length was selected to be within the attractive region of the LJ potential. Bonded units were treated as harmonic oscillators with the minimum close to the LJ potential minimum. The oscillator overlapping distance was shifted to the left of the overlapping distance of LJ potential. A rough sketch of placements of potentials are shown in the Figure 3.

To facilitate homogeneous growth process, oscillator potential minima was selected to be same as LJ potential minima, for most of our simulations. Any given displacement for bonded units was accepted if the move is towards the oscillator minimum. Otherwise it was rejected. Any given displacement of a bonded unit was

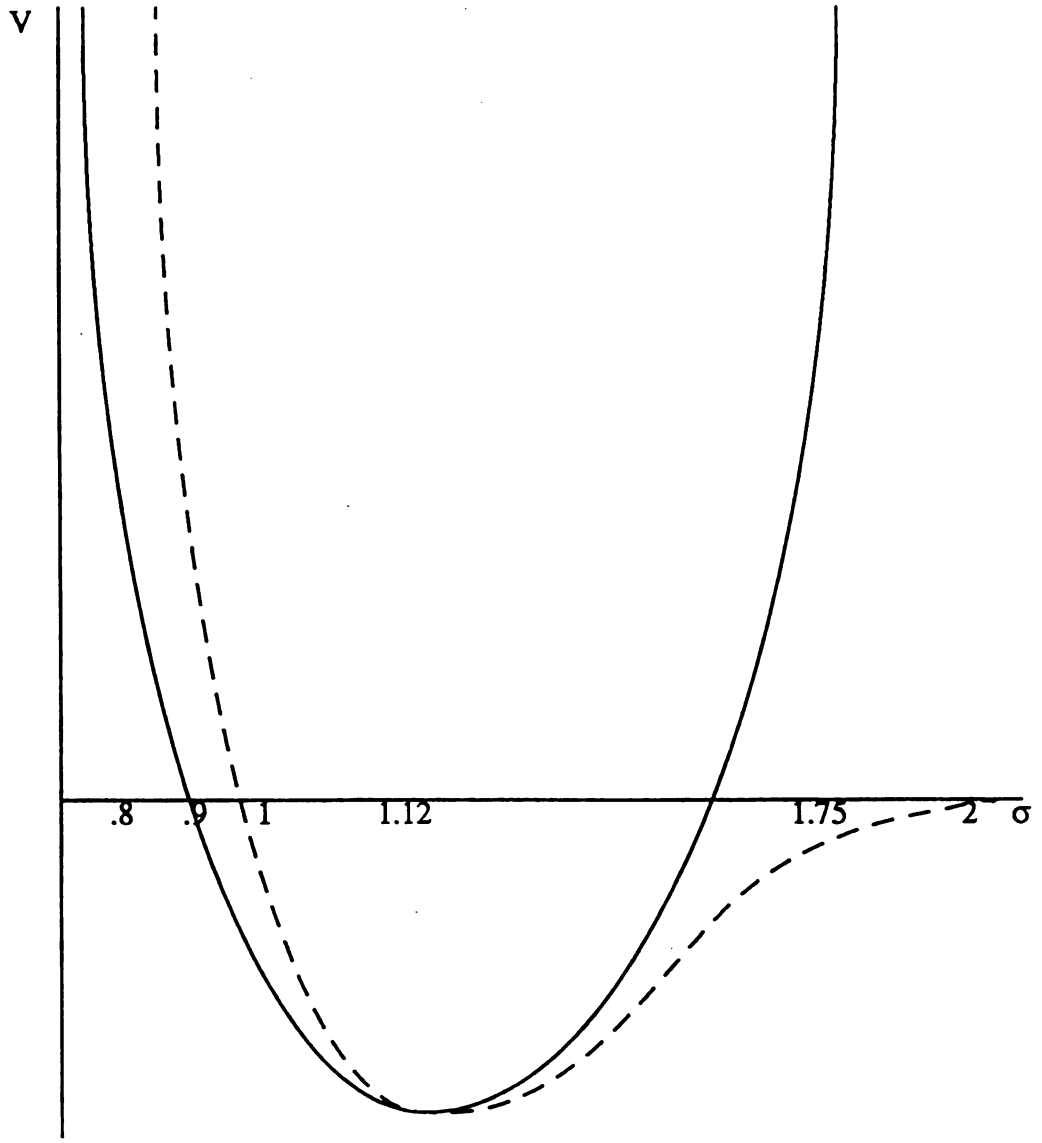


Figure 3. Rough Sketch of Oscillator and LJ Potential Placements.
LJ potential minimum = Oscillator potential minimum = 1.12σ
LJ potential overlap distance = 0.9σ
Oscillator potential overlap distance = 0.8σ
Reaction distance = Maximum bond length = 1.75σ
LJ long-range cutoff = 2.2σ
Reduced density = density $*(\sigma^3) = 0.64$
Reduced temperature = $k_B T/\varepsilon = 1.06$

immediately rejected if it results in an overlap with its bonded neighbor or the resulting bond length exceeds the reaction distance. A large reduction of bonded unit displacement will delay the move towards the oscillator minimum. This was concluded by calculating the fraction of bonds reached to their equilibrium distance at different stages of the given simulation. At different stages of the simulation the configuration was analyzed to calculate bond lengths. Necessary computer programs for this process is included in the appendix C.

As described in earlier paragraphs, to account for the reduced mobility of the polymer as compared to unreacted monomer, bonded units were given a reduced displacement in the simulations. This feature as well as the choice of the cut off of maximum allowed bond length (a technical choice) took steric hindrance into account and led to trapped active centers, but, in comparison to most earlier models, our trappings were not permanent, since the radical center could escape or a monomer could move within the vicinity of attractive potential after some time. Such trappings could have affected the gel point, if they occurred before the gel point was reached. At the gel point, enough chains were linked together to form a large macromolecular network that spanned the entire reaction system. With higher multiple center concentration, the cross-linking was complete, and an infinite network formed, before any trapping occurred. Our results showed that chain cross-linking was complete, before any trappings of the radicals, with reasonable concentration of multiple centers.

By having multiple centers which could enter into the same linear chain at a later stage of the growth, chain cyclization was allowed in our model. Some of the earlier models did not take this into consideration. Destruction of active centers by annihilation

was not allowed in our model, since we kept the concentration of active centers at a very low percentage. The probability of such an annihilation was considered to be negligible.

The major advantage of our technique over previously published work is that the molecular mobility of non-bonded, as well as bonded units, are allowed in a realistic manner during the polymer growth. Incorporation of realistic potentials during the growth is another advantage over earlier work by many authors. At the end of our simulation the bond lengths of the cross-linked polymer backbone are closer to their equilibrium values.

It is possible to turn on a torsional potential after this stage, and the system can be left to reach to conformational equilibrium. Alternatively, one can incorporate the torsional potential along with the oscillator potential and use energy minimization.

4.2. Simulation Results and Discussion

Nine hundred and thirty four monomer units within a simulation box with periodic boundary conditions were allowed to polymerize, according to above algorithm. Four initiators and required number of multiple centers were labeled randomly at the beginning. All our simulations were carried out at reduced LJ temperature of 1.06 and an initial reduced LJ density of 0.64. All parameters in our simulations used LJ diameter (σ) as the unit of length. Our monomers were assumed to be at least two c-c bond length long. The simulation box was of length 2 holding approximately 1000 monomer units evenly spaced in a cubic lattice at the beginning. The correspondence with the physical values can be related through these. LJ reduced temperature and initial reduced density was selected to assure the literature LJ potential values at the beginning.

One of the most significant contributions of our simulation results is an enhanced understanding of radical trapping. In earlier work by many authors, the simulation was based on a random walk on a cubic lattice and an active center was considered as trapped when all its nearest neighbors (on the lattice) were occupied [17]. As a result, these authors observe early trapping at a very low conversion, a very high fraction of trapped active centers and a relatively low limiting conversion at which all active centers are trapped. For example, simulations by Herrmann, Stauffer and Landau [17] have predicted a limiting conversion of 55% for a system in which experimental observations indicates conversion over 80% are readily achieved. Their predictions are shown in the Figure 4.

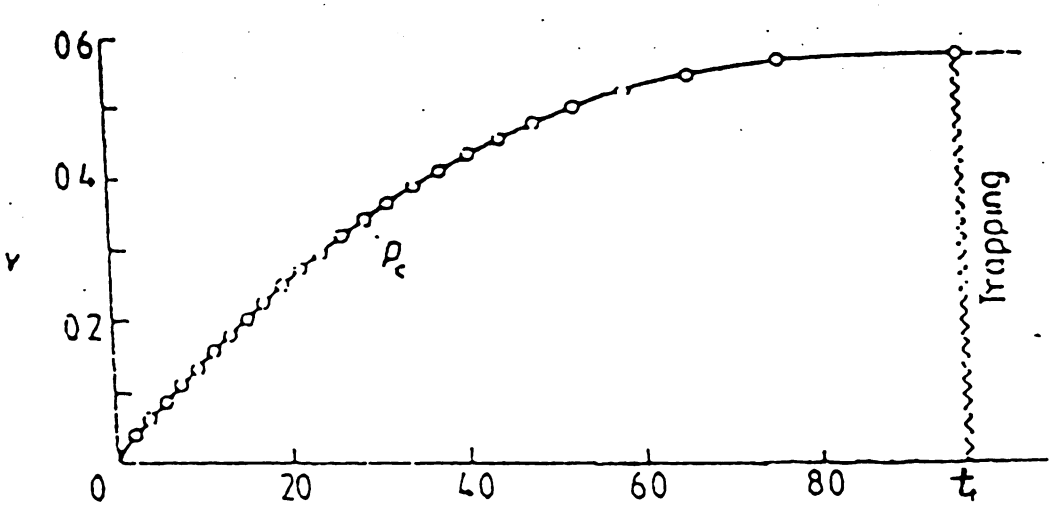


Figure 4. Conversion (v) against time for $C_i = 0.004$, $C_t = 0.1$ and L=42. The gel point is marked by p_c (From Herrmann, Stauffer and Landau [17]).

In addition, the simple kinetic gelation simulations predict that trapping increases as the fraction of tetra-functional monomer is decreased. These limitations ultimately arises from the fact that the simulations neglect molecular mobility altogether, and therefore active centers become trapped after propagating with their nearest neighbors.

In our work, we distinguished between two types of trapping mechanisms. The first kind was a temporary trapping due to the possibility that the closest neighbor to the active center was beyond the attractive region of the potential well. However, the situation might soon change due to the mobility of units. We observed that this type of early trapping did not usually occur with a reasonable choice of the reaction distance for bonding. Even if they occurred at early stage of the growth, they were only temporary and did not fall in the category of a permanent trapping. The second kind results from an active center being trapped inside a cage of bonded units. This trapped center usually did not get reactivated again due to steric hindrance. This was more permanent in nature and usually occurred at a latter part of the growth process.

Typical results from our Monte Carlo simulations are shown in Figure 5. As in all the simulations in this thesis 934 monomer units were contained in a simulation box with four propagating active centers. The specific conditions for each simulation are listed at the top of each Figure. Examination of Figure 5 reveals that the initial portion of the conversion against Monte Carlo time (cycles) profile is linear. This arises from the fact that all four active centers propagate each Monte Carlo cycle since there is no trapping. After 155 MC cycles the first active center became permanently trapped and the slopeof the profile exhibits a discontinuous change from 0.428% / MC cycle to 0.3219% / MC cycle. Similarly, the second and third active centers became trapped after 174

and 200 MC cycles, respectively, with corresponding shifts in the slope of the conversion against time profile. Our simulation ended when the final active center was trapped.

In contrast to previous simulations (i.e. Figure 4), the results shown in Figure 2 correspond to a relatively high limiting conversion approaching 80%. At the limiting conversion, all active centers become trapped and no further reaction is possible. Previous simulations overestimate trapping since molecular mobility is neglected. In our simulations, the value of the limiting conversion is sensitive to the reaction distance parameter, as illustrated in Figures 5-7. These Figures illustrate that as the reaction distance is decreased, trapping occurs at a lower conversion, and the limiting conversion is decreased. As shown in Figure 7 in our work a limiting conversion about 60% can be achieved with a reaction distance of 1.6σ. With a reaction distance of 1.7σ, it was shown in our work that all active centers did not get trapped until 72% conversion. One thing to notice is that even the distance 1.75σ is well within the attractive region of the LJ potential well. With 1.75σ as the reaction distance, it was possible to reach up to 80% limiting conversion, consistent with experimental observations.

Figure 8 illustrates the effect of tetra-functional groups on cross-linking. Higher percentage of tetra-functional groups in general will lead to more cross-linking within the same number of MC cycles. After 140 MC cycles sample with 30% tetra-functional groups has formed 52 cross-links whereas sample with 20% tetra-functional groups has formed 38 cross-links. Figure 9 (30% tetra-functional groups) illustrates a typical relationship between the total number of cross-links and chain cyclizations. Within first 140 MC cycles there were 52 total cross-links with 20 cyclizations. Probabilities of cross-linking and chain cyclizations can be obtained by this technique.

R

P

L
C
C
C

% Conversion

Run number 1:

Parameters for potentials

LJ overlapping distance	$= 0.9\sigma$
Oscillator overlapping distance	$= 0.6\sigma$
Oscillator maximum distance	$= 1.75\sigma$
Oscillator equilibrium distance	$= 1.12\sigma$

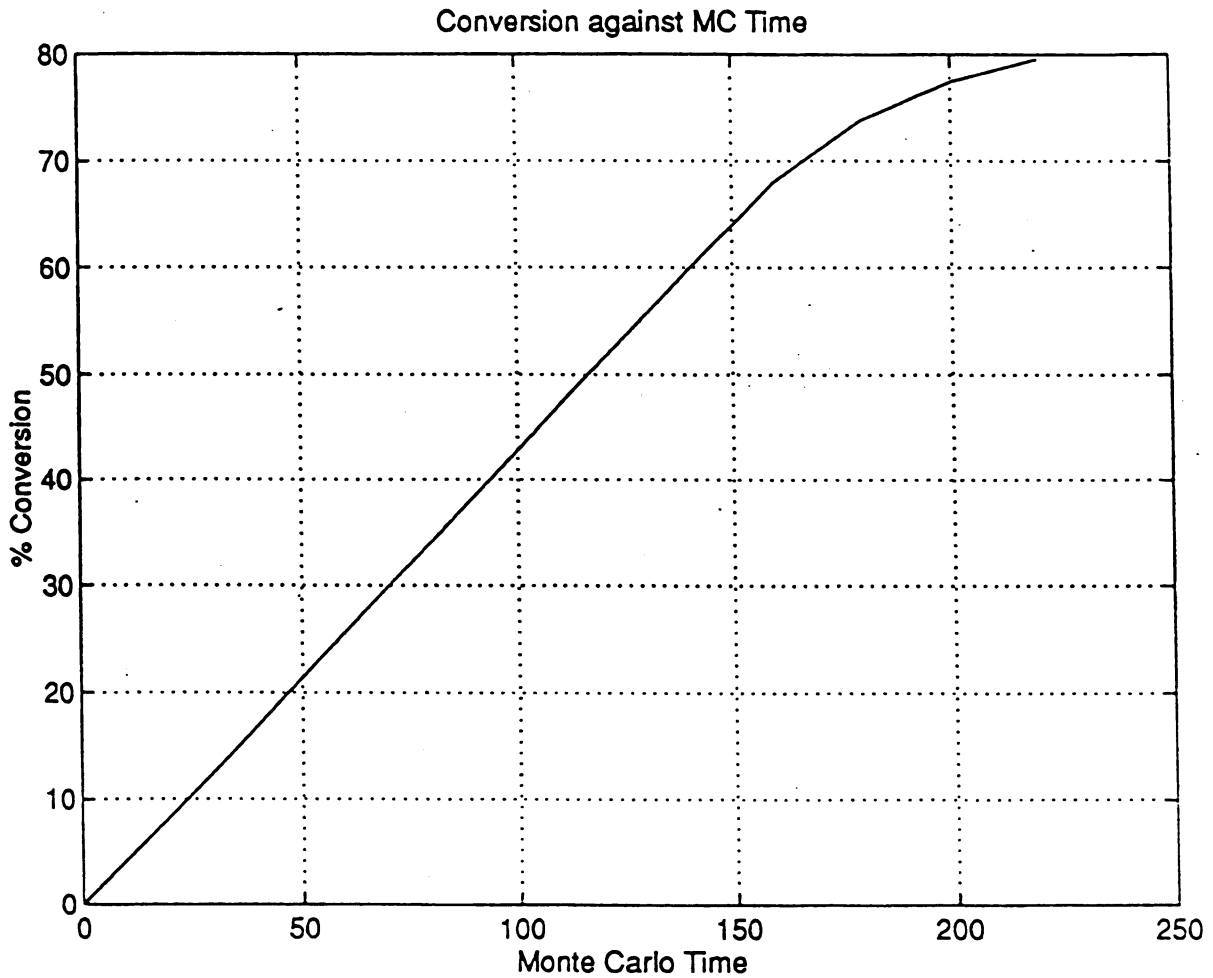


Figure 5. Conversion against MC time for a reaction distance of 1.75σ. $C_i = 0.004$, $C_t = 0.2$, 80% reduction of polymer displacement (with further reduction by a factor of 0.995 between each MC cycle).
[C_i = initiator concentration, C_t= tetra-functional unit concentration]

Re

Pa

1000

8

7

5

% Conversion

Run number 2:

Parameters for potentials

LJ overlapping distance	$= 0.9\sigma$
Oscillator overlapping distance	$= 0.6\sigma$
Oscillator maximum distance	$= 1.7\sigma$
Oscillator equilibrium distance	$= 1.12\sigma$

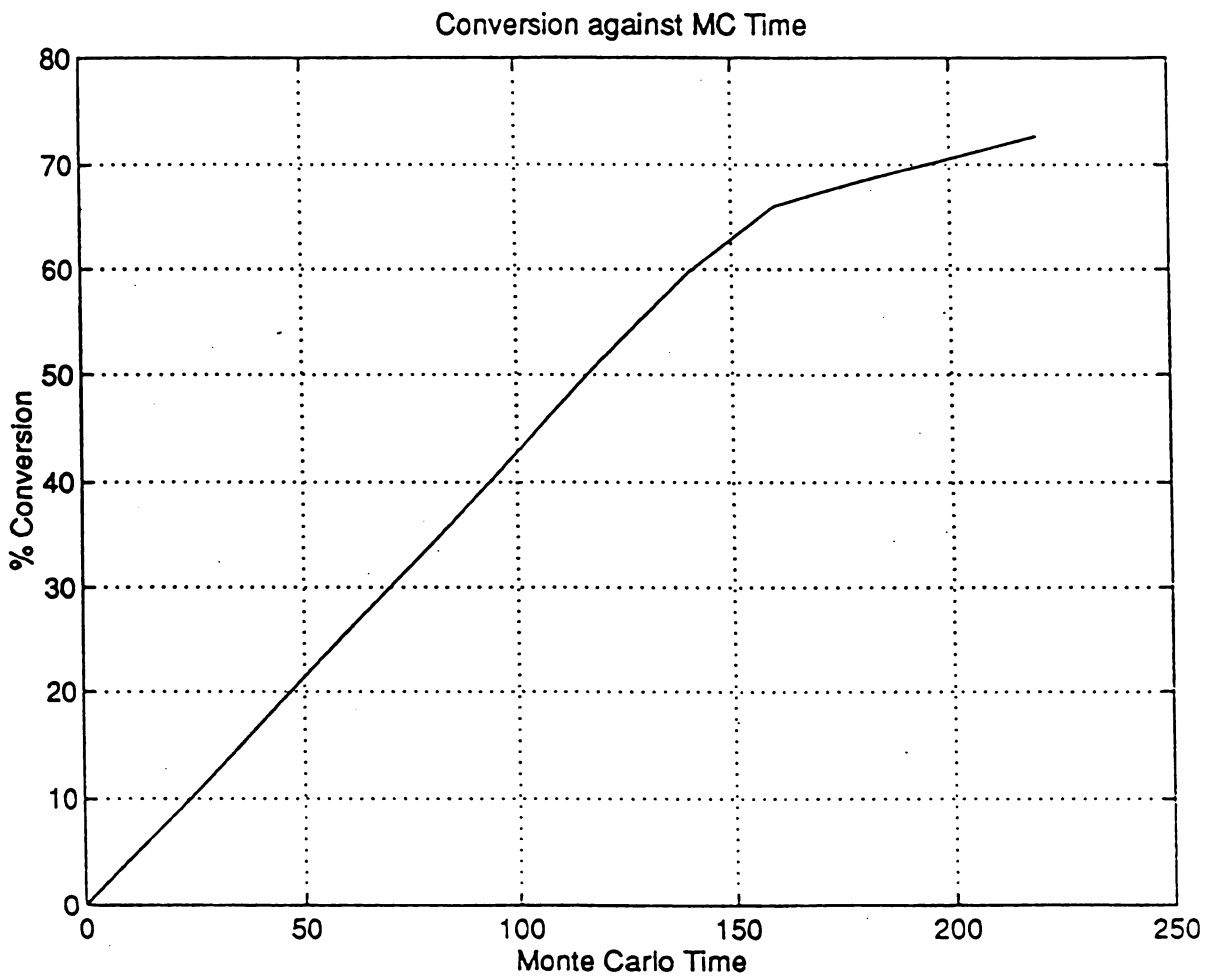


Figure 6. Conversion against MC time for a reaction distance of 1.7σ. $C_i = 0.004$, $C_t = 0.2$, 80% reduction of polymer displacement (with further reduction by a factor of 0.995 between each MC cycle).
[C_i = initiator concentration, C_t= tetra-functional unit concentration]

% Conversion

Run number 3:

Parameters for potentials

LJ overlapping distance	$= 0.9\sigma$
Oscillator overlapping distance	$= 0.6\sigma$
Oscillator maximum distance	$= 1.6\sigma$
Oscillator equilibrium distance	$= 1.12\sigma$

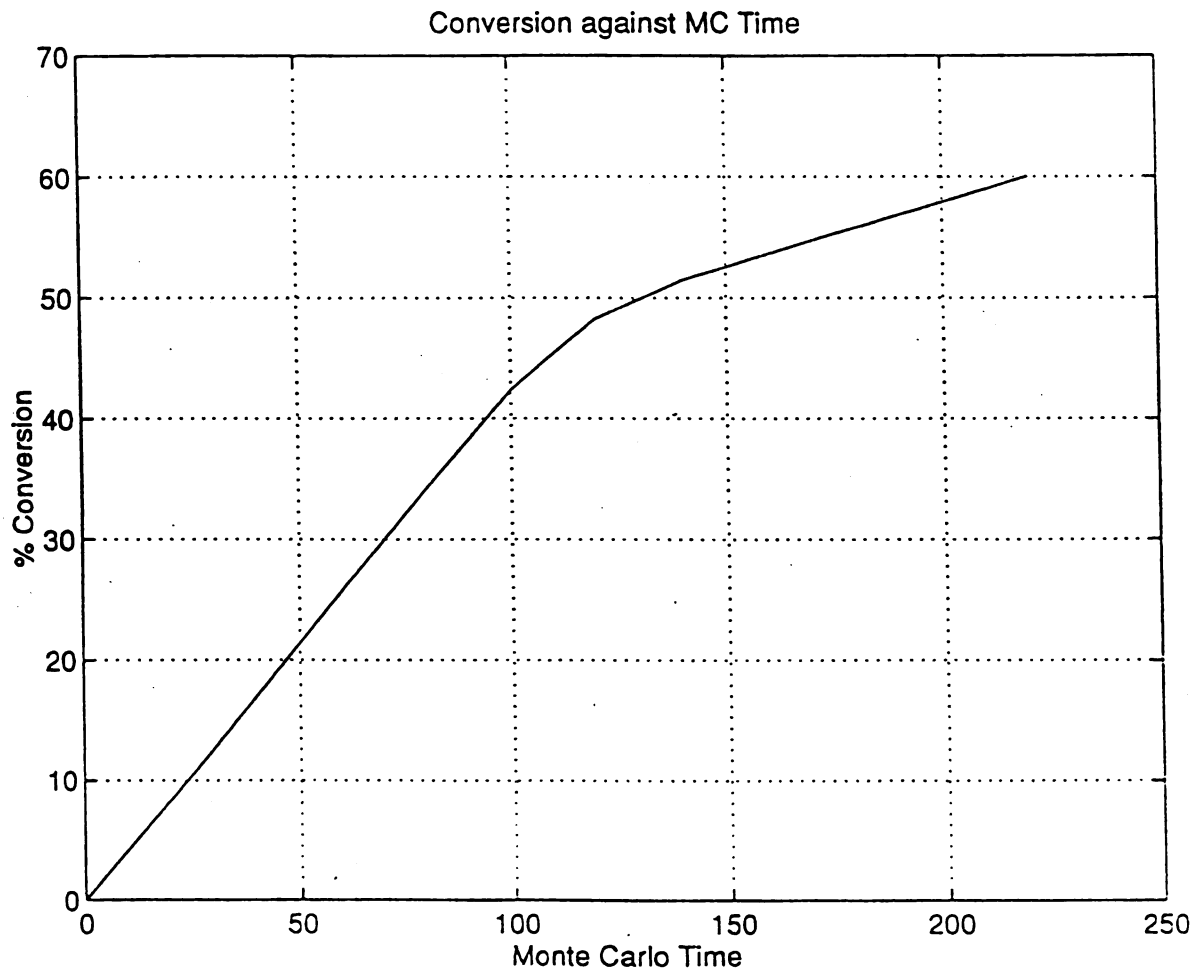


Figure 7. Conversion against MC time for a reaction distance of 1.6σ. $C_i = 0.004$, $C_t = 0.2$, 80% reduction of polymer displacement (with further reduction by a factor of 0.995 between each MC cycle).
[C_i = initiator concentration, C_t= tetra-functional unit concentration]

A

Pa

L
0
0
0

Number of Cross-links

Analysis of cross-linking and cyclizations:

Parameters for potentials

LJ overlapping distance	$= 0.9\sigma$
Oscillator overlapping distance	$= 0.6\sigma$
Oscillator maximum distance	$= 1.75\sigma$
Oscillator equilibrium distance	$= 1.12\sigma$

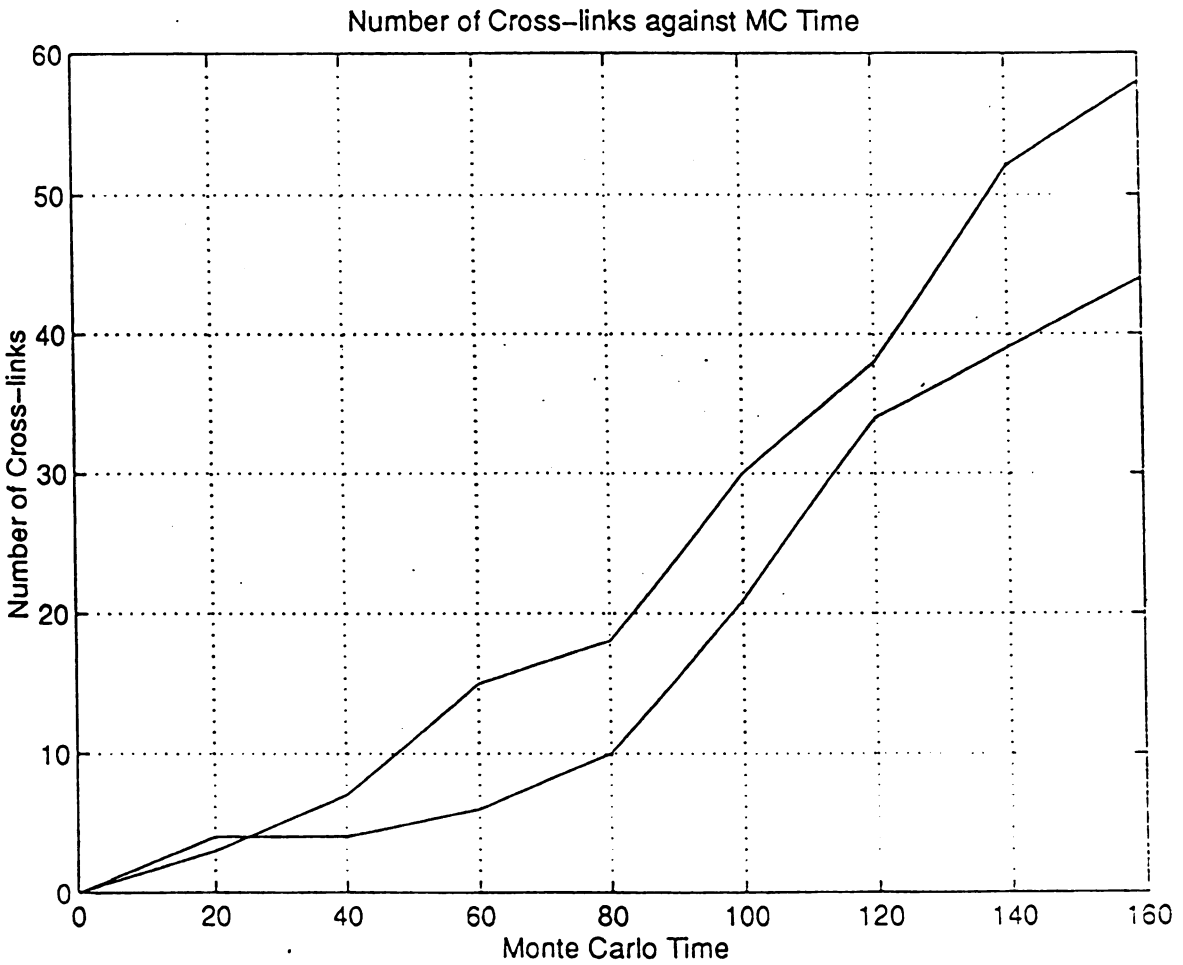


Figure 8. Total number of cross-links against MC time for $C_i = 0.004$, $C_t = 0.2$ (lower curve). $C_i = 0.004$, $C_t = 0.3$ (upper curve). 80% reduction of polymer displacement (with further reduction by a factor of 0.995 between each MC cycle). [C_i = initiator concentration, C_t= tetra-functional unit concentration]

Number of Cross-links _ , Number of Cyclizations _

Parameters for potentials

LJ overlapping distance	$= 0.9\sigma$
Oscillator overlapping distance	$= 0.6\sigma$
Oscillator maximum distance	$= 1.75\sigma$
Oscillator equilibrium distance	$= 1.12\sigma$

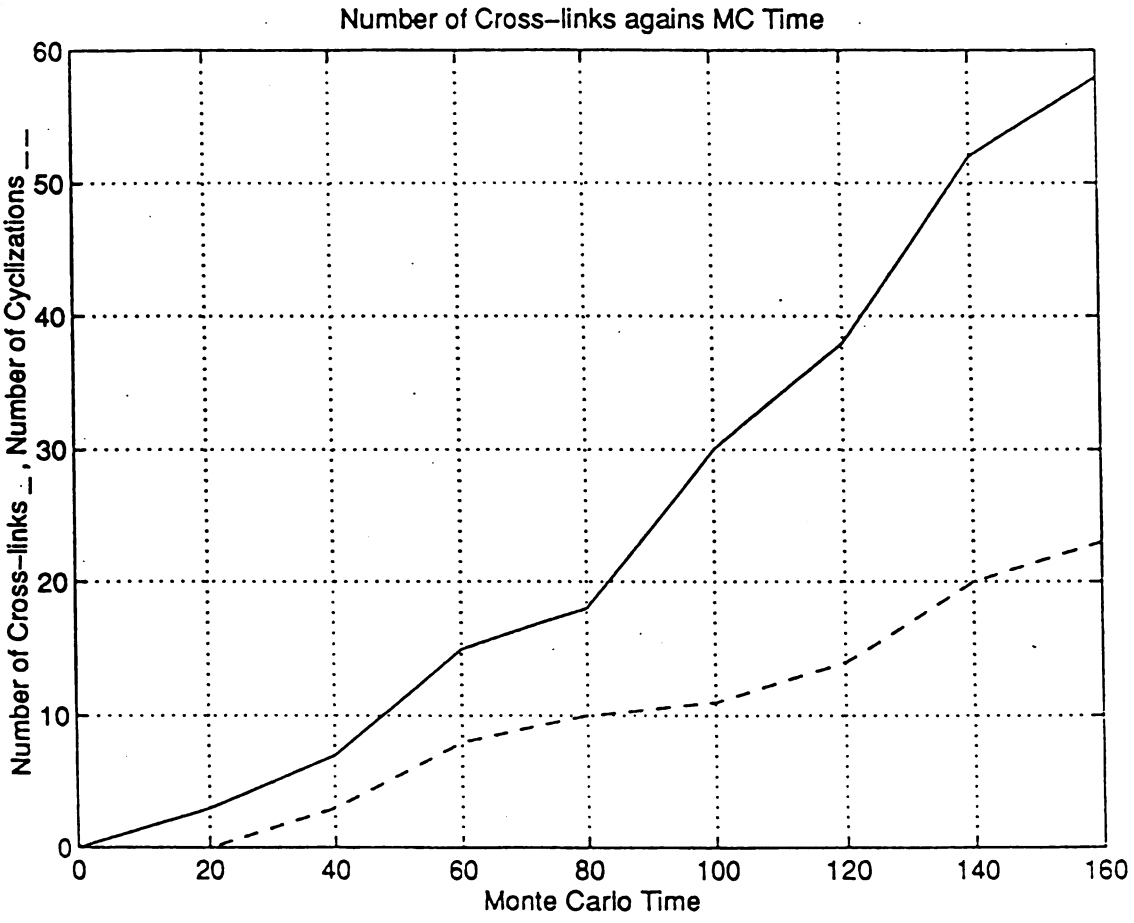


Figure 9. Total number of cross-links against MC time for $C_i = 0.004$, $C_t = 0.3$ (- curve). Number of chain cyclizations against MC time (- - curve). 80% reduction of polymer displacement (with further reduction by a factor of 0.995 between each MC cycle).
[C_i = initiator concentration, C_t= tetra-functional unit concentration]

Conversion against MC time plots obtained at the end of the simulation for different variable parameters, such as maximum bond length and different degrees of polymer mobilities are included in previous pages. An illustration of a variation of cross-linking density with the concentration variation of tetra-functional units is also included. The complete computer program for this process is included in appendix B and corresponding flowcharts are included in appendix A.

If desired, one can shift the oscillator potential away from the LJ potential, but such an act could lead to density inhomogeneties within the system. We believe the above difficulty could be avoided by using a scheme which employs a reduced hard-sphere diameter at high density or using a soft (inverse ninth through inverse fourth power) repulsion [28]. In reference 28, a perturbation method was reported which divides the pair potential into a reference potential and a perturbation potential at a break point which may depend on density. A better method to avoid this problem is to work in a constant NPT-ensemble.

5. A NEW MODEL FOR POLYMERIC GLASS STRUCTURES

With some modification, the same computer code that we developed to study gelation can be used to generate long chain alkanes. In this work monomer units were considered as -CH_2 groups. The oscillator equilibrium length was brought towards the C-C bond length. The shift of oscillator potential minimum will change the density of the system during the growth. The corresponding change in volume can be handled in NPT-ensemble. In NVT-ensemble, replacement of stiff inverse twelfth power potential with a soft inverse ninth power repulsion [28] might be more useful here. In contrast to Gupta, Weistermann, and Bitsanis [27] model, growth and equilibrium did not compete with each other in our model, rather it was simultaneous throughout the growth (In reference 27 a decision was made on wether to further grow the chain or to attempt an equilibration move). As in their model lack of internal rotations was a disadvantage in ours too. With some modifications in our computer program, number of chains growing within the system can be increased.

There were two major advantages in our method over that of Khare, Paulaitis and Lustig [26] method. In addition to realistic mobility of particles, bond lengths reached to their equilibrium values simultaneously with the growth. A complete relaxation can be achieved by an incorporation of a torsional potential along with the harmonic oscillator potential. With some modifications, our model will provide a simplified, realistic and computationally fast way to study polymeric glass structures and long chain alkane growth.

6. SUMMARY AND CONCLUSIONS

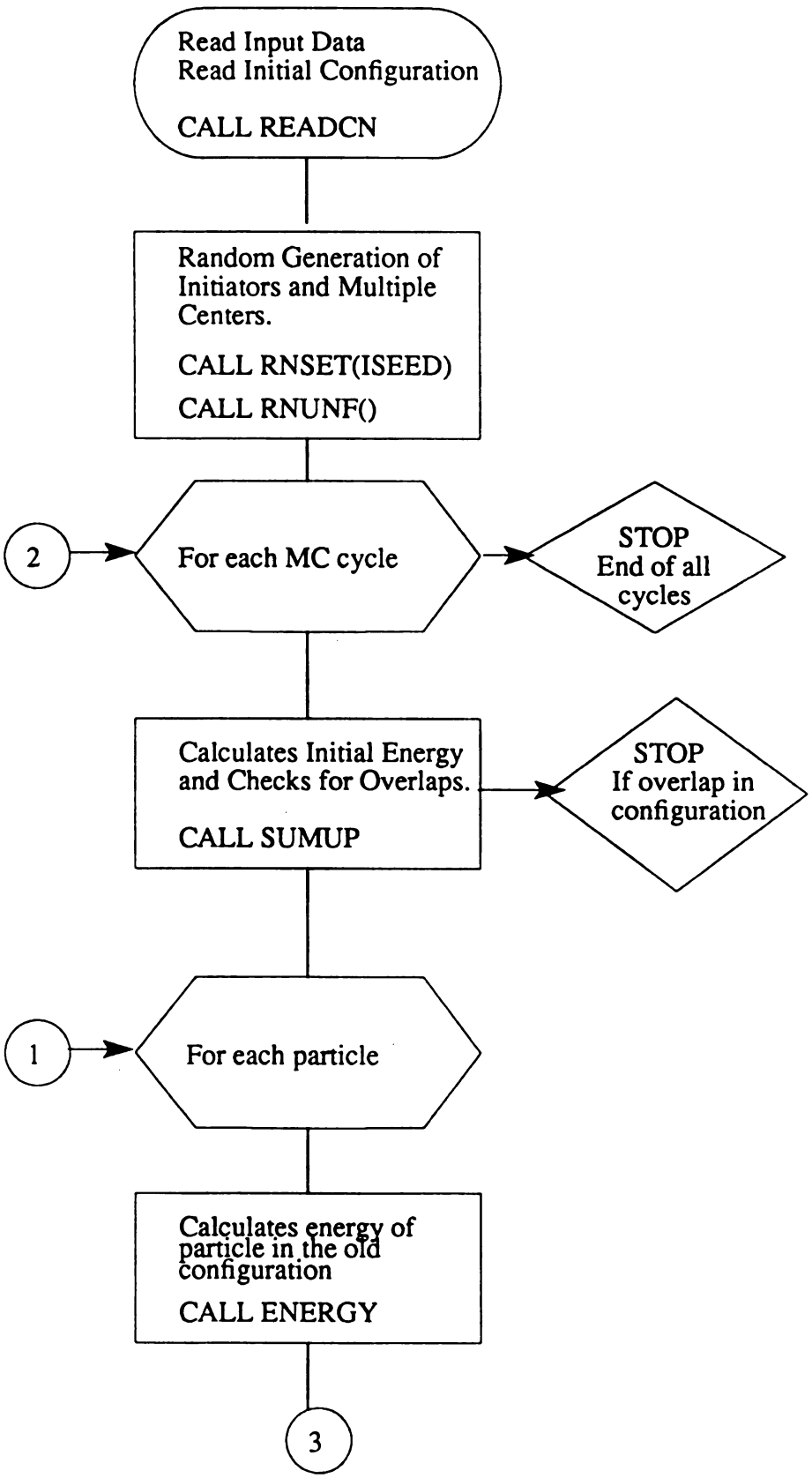
We have proposed a new realistic model for kinetic gelation studies, and its computational feasibility was studied. The importance of incorporating molecular mobility and realistic potentials in the study of cross-linking polymerization process was shown, using Monte Carlo simulations.

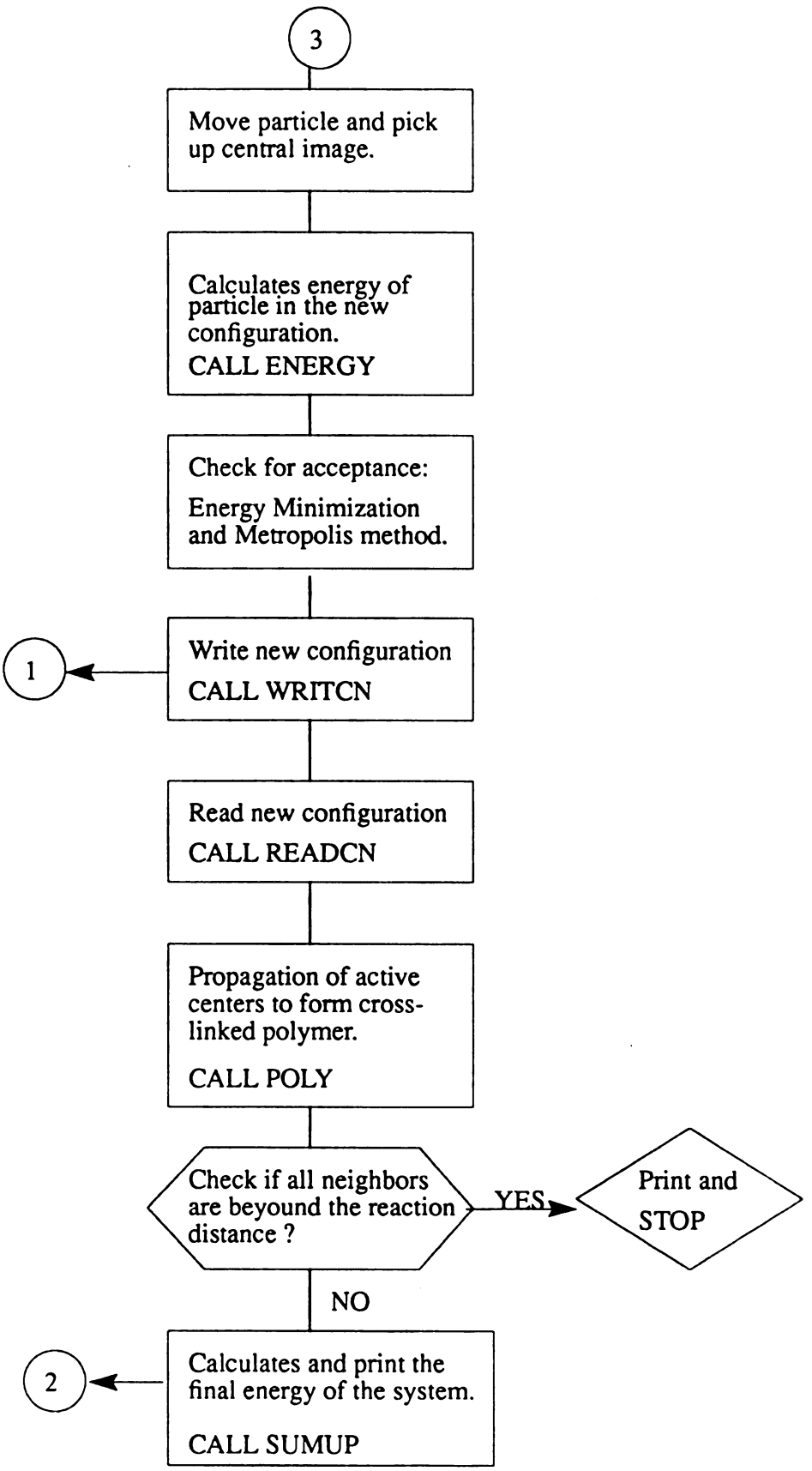
In contrast to most earlier simulation methods for gelation, we did not fix the monomer units on a lattice but allowed the monomer and polymer units to move in a physically realistic manner. Another important modification was to take into account, both intra- and intermolecular potentials during the growth and the simultaneous equilibration process. Polymer mobility and the potential coordinates were carefully chosen, in order to avoid possible density inhomogenities that could build up within the growth and equilibration. The importance of this factor in gelation compared to that of long chain alkane growth was realized. An added advantage of the employment of realistic potentials during the process was to give some insight into internal architecture of the polymer backbone. Growth and the equilibration were simultaneous and did not compete with each other in our model. Chain cyclization was incorporated, and it was shown that the possibility of an early trapping was considerably reduced by taking molecular mobility in to account. Chain cyclization was an another ignored aspect of earlier gelation models. Cage trapping occurred at the latter part of the growth, and the ratio of percentage of trapped centers to that of active centers remained low until the system reaches a very high conversion. Though reactivation of trapped active centers was possible within our model, in most cases trapping occurred after the formation of gel phase in the presence of reasonable concentration of cross-linking agents.

APPENDICES

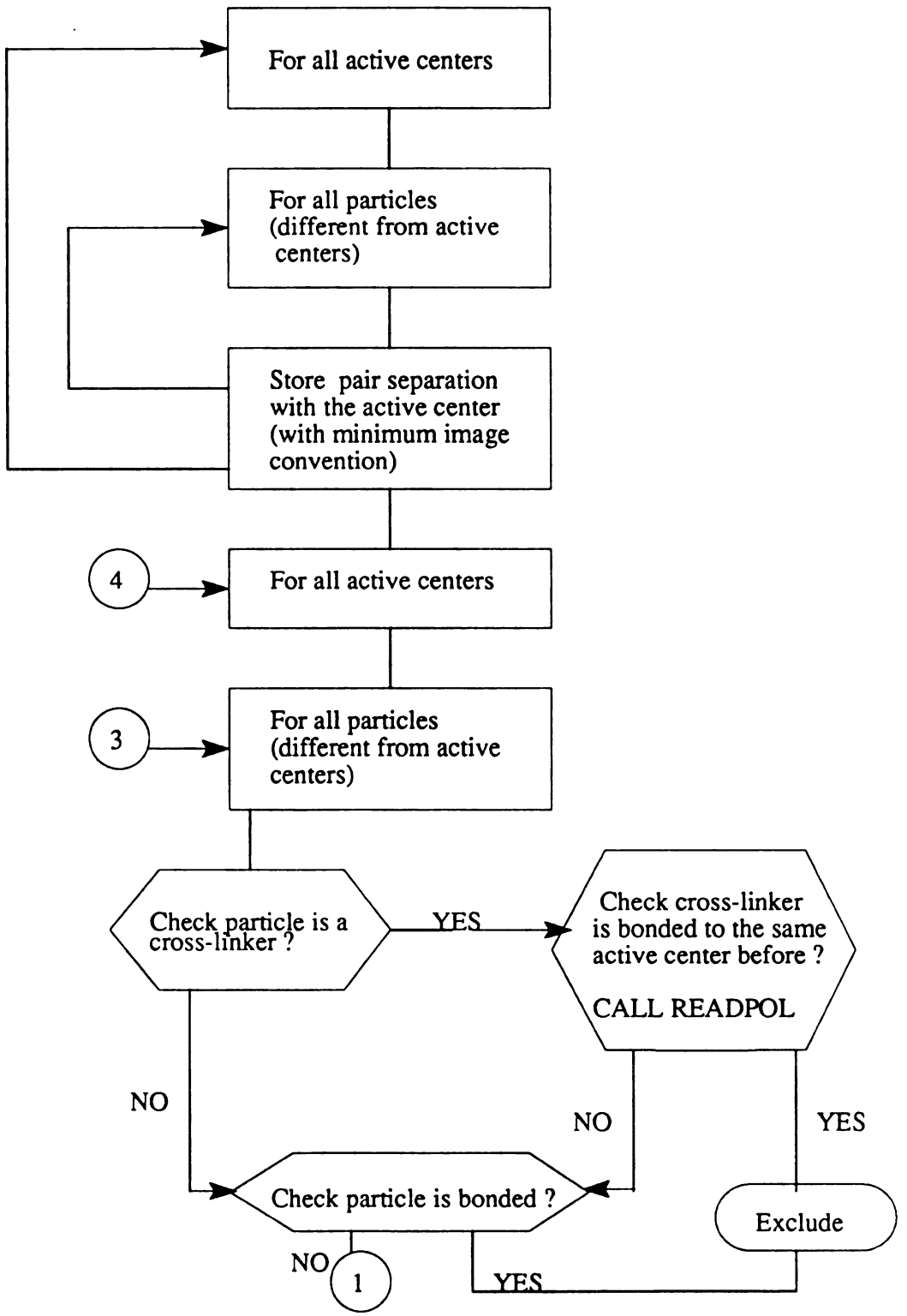
APPENDIX A

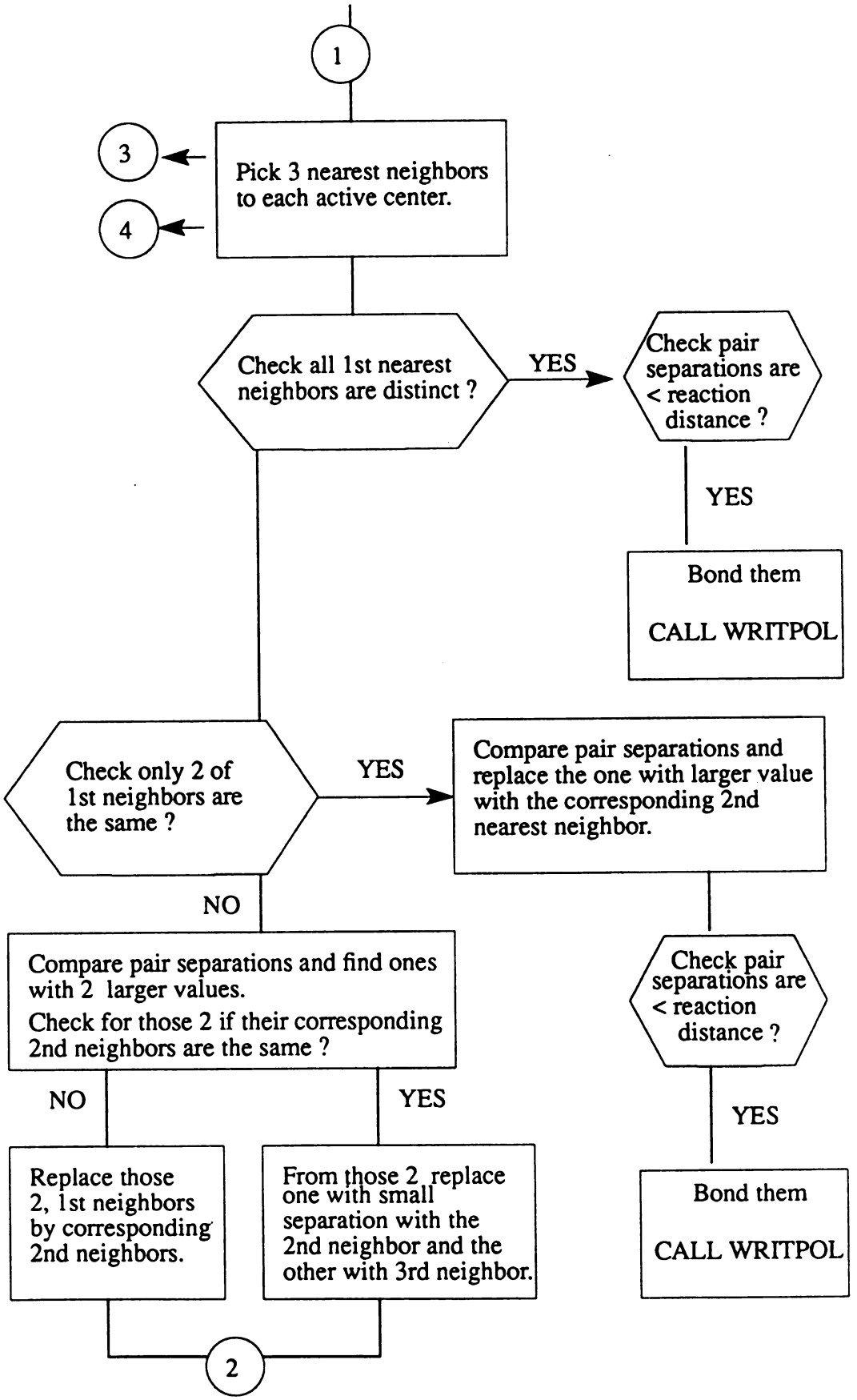
FLOW CHART FOR THE MAIN PROGRAM

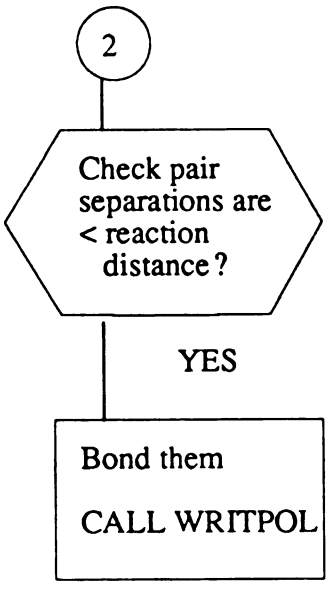



3
Move particle and pick up central image.
Calculates energy of particle in the new configuration.
CALL ENERGY
Check for acceptance:
Energy Minimization and Metropolis method.
1
Write new configuration
CALL WRITCN
Read new configuration
CALL READCN
Propagation of active centers to form cross-linked polymer.
CALL POLY
Check if all neighbors are beyound the reaction distance ?
YES
Print and STOP
NO
2
Calculates and print the final energy of the system.
CALL SUMUP

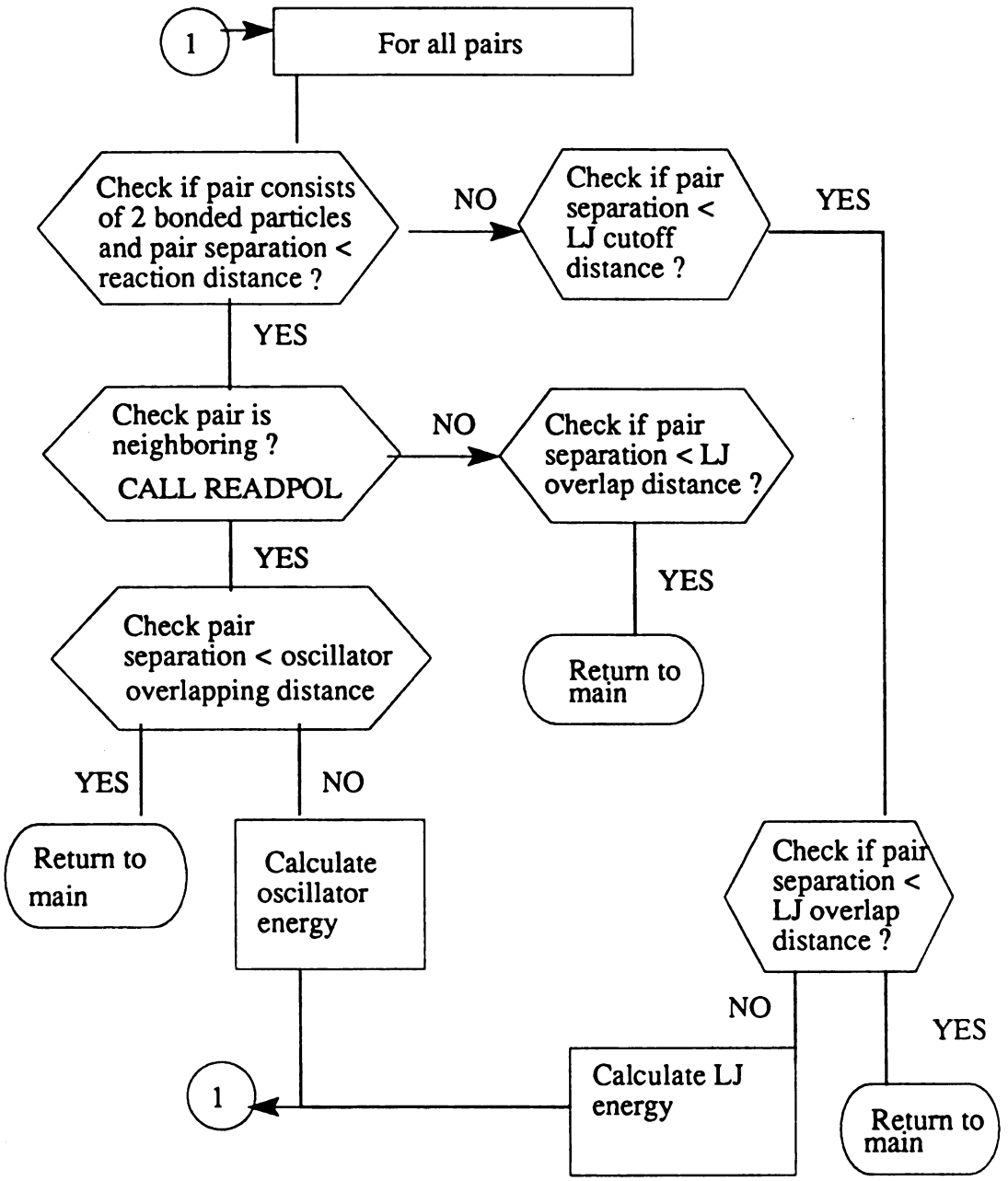
FLOW CHART FOR THE POLYMER SUBROUTINE



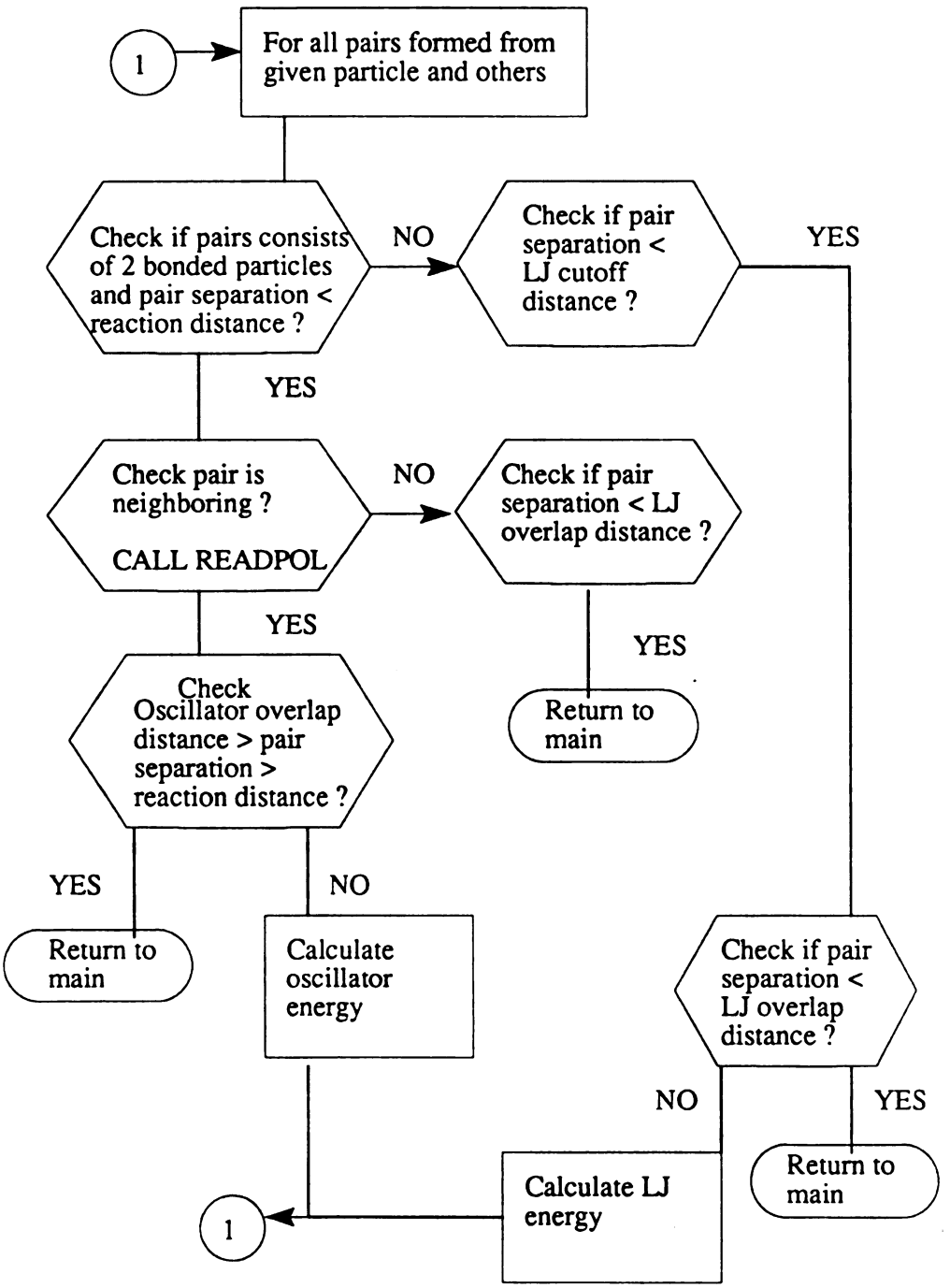

1
3
4
Pick 3 nearest neighbors to each active center.
Check all 1st nearest neighbors are distinct ?
YES
Check pair separations are < reaction distance ?
YES
Bond them
CALL WRITPOL
Check only 2 of 1st neighbors are the same ?
YES
Compare pair separations and replace the one with larger value with the corresponding 2nd nearest neighbor.
NO
Compare pair separations and find ones with 2 larger values.
Check for those 2 if their corresponding 2nd neighbors are the same ?
Check pair separations are < reaction distance ?
NO
YES
YES
Replace those 2, 1st neighbors by corresponding 2nd neighbors.
From those 2 replace one with small separation with the 2nd neighbor and the other with 3rd neighbor.
Bond them
CALL WRITPOL
2


2
Check pair separations are < reaction distance?
YES
Bond them
CALL WRITPOL

FLOW CHART FOR OVERLAP CHECK SUBROUTINE



FLOW CHART FOR SUBROUTINE TO CALCULATE ENERGY



APPENDIX B

```
c*****************************************************************
c  ** Constant-nvt monte carlo program for hard sphers          **
c  ** with a routine to form a crosslinked polymer.             **
c  **                                                           **
c  ** Program can be use for gel point calculations and        **
c  ** molecular weight distribution evaluations.               **
c  **                                                           **
c  ** Molecular mobility is allowed towards the energy         **
c  ** minimum throughout the polymerization. Mobility of       **
c  ** bonded units were reduced with the growth. This          **
c  ** is limited by the approach to oscillator equilibrium.    **
c  *****************************************************************
c
c  *****************************************************************
c  ** Program can be used for any given number of monomers     **
c  ** with an appropiate coordinate file and minor changes.    **
c  **                                                           **
c  ** Number of initiaters are resticted to four in polymer    **
c  ** routine, but can be change to any number with an         **
c  ** appropiate modifications.                                **
c  *****************************************************************

      program ljgel
      common /block1/ rx,ry,rz

c  *****************************************************************
c  ** The box is of unit length, -0.5 to +0.5                   **
c  ** Box length can be changed with an appropiate change in    **
c  ** periodic boundary conditions.                             **
c  **                                                           **
c  ** Principal variables:                                      **
c  **                                                           **
c  ** integer n                  number of monomers             **
c  ** integer nstep              maximum number of cycles       **
c  ** integer iprint             print interval                 **
c  ** integer isave              save interval                  **
c  ** integer iratio             max displacement update        **
c  **                                          interval        **
c  ** real    rx(n),ry(n),rz(n)  positions                      **
c  ** real    dens               reduced density                **
c  ** real    sigma              hard sphere (LJ) diameter      **
c  ** real    drmax              reduced maximum displacement   **
c  ** real    temp               reduced temperature            **
c  ** real    rmin               minimum reduced pair           **
c  **                                          separation      **
c  ** real    rcut               reduced cutoff distance        **
c  ** real    nbond              maximum number of bonds in a   **
c  **                                    given linear chain    **
c  ** real    bonle              maximum bondlength             **
c  ** real    eql                equlibrium bondlength          **
c  ** real    foc                force constant                 **
```

```
c  ** real    tvbmax          maximum Bolzman factor        **
c  ** real    control         energy diffrence control      **
c  ** real    factr           max displacement reduction    **
c  ** real    v               the potential energy          **
c  ** real    voscil          the oscilator energy          **
c  ** real    red             multiplicative factor to      **
c  **                         adjust rmin for the oscilator **
c  ** logical ovrlap          true if 2 units overlap       **
c  ** logical ovrpol          true if 2 bonded units overlap**
c  **                         after the displacement        **
c  ** logical touch           true if others overlap        **
c  ** logical neig            true if 2 bonded units are    **
c  **                                              neighbors **
c  ** logical flip            true if all possible bonds    **
c  **                         were formed before the end of **
c  **                         all cycles                    **
c  **                                                       **
c  ** Units:                                                **
c  **                                                       **
c  ** The program uses Lennard-Jones units for user input   **
c  ** and ouput but conducts the simulation in a box of     **
c  ** unit length.                                          **
c  ** For example, for a boxlength L, and Lennard-Jones     **
c  ** parameters epsilon and sigma, the units are:          **
c  **                                                       **
c  ** Property       LJ Units          Program Units        **
c  **                                                       **
c  ** temp           epsilon/K         epsilon/K            **
c  ** pres           epsilon/sigma**3  epsilon/L**3         **
c  ** v              epsilon           epsilon              **
c  ** dens           1/sigma**3        1/L**3               **
c  **                                                       **
c  ** Routines referenced:                                  **
c  **                                                       **
c  ** Subroutine sumup                                      **
c  **   Calculates the potential energy for a configuration **
c  ** Subroutine energy                                     **
c  **   Calculates the potential energy of atom i with all  **
c  **   the other atoms in the liquid. For bonded neighbors **
c  **   the oscillator energy is evaluated                  **
c  ** Subroutine poly                                       **
c  **   Forms the crossliked polymer as the simulation      **
c  **   proceeds                                            **
c  ** Subroutine readcn                                     **
c  **   Reads in a configuration                            **
c  ** Subroutine writcn                                     **
c  **   Write out a configuration                           **
c  ** Subroutine readpol                                    **
c  **   Read the polymer file                               **
c  ** Subroutine writpol                                    **
c  **   Write out polymer file                              **
c  ***********************************************************
```

```
      integer n,i2(1,4),jcoun,ij(10)
      parameter (n=100)
      real rx(n),ry(n),rz(n)
      real drmax,dens,temp,sigma,rcut,beta,eql,foc
      real acm,accept,pi,ratio,rmin,tvbmax,vos,red
      real v,vnew,vold,vlast,vn,deltv,deltvb,vs,voscil
      real vlrc,vlrc6,vlrc12,control,factr,bonle
      real rxiold,ryiold,rziold,rxinew,ryinew,rzinew
      real voso,vosn,delvos
      real vosi,vosf
      integer step,i,nstep,iprint,isave,iratio
      integer iflag(100),nbond
      integer iseed
      external rnset,rnunf
      logical ovrlap,flip,ovrpol,neig
      character cnfile*10,polymer*10
      parameter (pi=3.1415927)
c     *****read input data*****************************************
      open (11,file='inputdata',status='old',form='formatted')
c     *****write input data****************************************
      read(11,'(i10 )')nstep
      read(11,'(i10 )')nbond
      read(11,'(f10.4)')red
      read(11,'(f10.4)')bonle
      read(11,'(f10.4)')eql
      read(11,'(f10.4)')foc
      read(11,'(i10 )')iprint
      read(11,'(i10 )')isave
      read(11,'(i10 )')iratio
      read(11,'(f10.4 )')temp
      read(11,'(f10.4 )')dens
      read(11,'(f10.4 )')rmin
      read(11,'(f10.4 )')drmax
      read(11,'(f10.4 )')tvbmax
      read(11,'(f10.4 )')rcut
      read(11,'(e12.4 )')control
      read(11,'(a )')cnfile
      read(11,'(a )')polymer
      write(*,'('' end of inputdata '')')
c     *****read initial configuration*****************************
      call readcn (cnfile)
c     *****convert input data to program units********************
      beta =1.0/temp
      sigma =(dens/real(n))**(1.0/3.0)
      rmin =rmin*sigma
      rcut=rcut*sigma
      bonle=bonle*sigma
      eql=eql*sigma
      drmax =drmax*sigma
      denslj=dens
      dens=dens/(sigma**3)
      if(rcut.gt.0.5)stop' cutoff too largr '
      acv=0.0
      acvsq=0.0
```

```
      flv=0.0
      acm=0.0
      accept=0.0
      factr=0.35
      do 1 jcoun=1,100
      iflag(jcoun)=1
    1 continue
c   *****calculate long range corrections**********************
      sr3 = (sigma/rcut)**3
      sr9 = sr3**3
      vlrc12 = 8.0*pi*denslj*real(n)*sr9/9.0
      vlrc6  = 8.0*pi*denslj*real(n)*sr3/3.0
      vlrc = vlrc12-vlrc6
c   ****write out information***********************************
      write(*,'(''sigma/box  = '',f10.4)')sigma
      write(*,'(''rmin/box   = '',f10.4)')rmin
      write(*,'(''rcut/box   = '',f10.4)')rcut
      write(*,'(''bonle/box  = '',f10.4)')bonle
      write(*,'(''eql/box    = '',f10.4)')eql
      write(*,'(''drmax/box  = '',f10.4)')drmax
      write(*,'(''tvbmax/box = '',f10.4)')tvbmax
      write(*,'(''control    = '',e12.4)')control
      write(*,'(''lrc for V  = '',f10.4)')vlrc
c   ****select initioators and multiple centers*****************
      iseed=123457
      call rnset(iseed)
      i2(1,1)=1+int(n*rnunf())
      i2(1,2)=1+int(n*rnunf())
      i2(1,3)=1+int(n*rnunf())
      i2(1,4)=1+int(n*rnunf())
      do 2 l=1,4
      iflag(i2(1,l))=0
    2 continue
      do 3 l=1,10
      ij(l)=1+int(n*rnunf())
      iflag(ij(l))=3
    3 continue
      vos=0.0
c   ****loops over requied bond formation per linear chain*******
      do 102 k=1,25
c   ****calculates initial energy and checks for overlap*********
      call sumup(rcut,rmin,sigma,ovrlap,v,iflag,eql,foc,vosi,
     +          polymer,neig,bonle,red)
      if(ovrlap)stop 'ovrlap in initial con '
      vs=(v+vlrc)/real(n)
      write(*,'(''initial v   = '',e20.6 )') vs
      write(*,'(/'' start of markov chain  ''/)')
      write(*,'(''  nmove        ratio        v/n    ''/)')
c   *****loops over all cycles and all molecules*****************
      do 100 step=1,nstep
            do 99 i=1,n
            rxiold=rx(i)
            ryiold=ry(i)
            rziold=rz(i)
```

```
c  **** calculates the  energy of i in the old configuration****
     call energy(rxiold,ryiold,rziold,i,rcut,sigma,vold,
    +     iflag,eql,foc,voso,ovrpol,rmin,polymer,neig,bonle,red)
c  *****move i and pickup the central image*********************
     if (iflag(i).eq.0.or.iflag(i).eq.2) then
     drmax1=drmax*factr
     rxinew=rxiold+(2.0*rnunf()-1.0)*drmax1
     ryinew=ryiold+(2.0*rnunf()-1.0)*drmax1
     rzinew=rziold+(2.0*rnunf()-1.0)*drmax1
     else
     rxinew=rxiold+(2.0*rnunf()-1.0)*drmax
     ryinew=ryiold+(2.0*rnunf()-1.0)*drmax
     rzinew=rziold+(2.0*rnunf()-1.0)*drmax
     end if
     if (rxinew.gt.0.5) then
         rxinew=rxinew-1.0
     else if (rxinew.lt.-0.5) then
         rxinew=rxinew+1.0
     end if
     if (ryinew.gt.0.5) then
     ryinew=ryinew-1.0
     else if (ryinew.lt.-0.5) then
     ryinew=ryinew+1.0
     end if
     if (rzinew.gt.0.5) then
         rzinew=rzinew-1.0
     else if (rzinew.lt.-0.5) then
         rzinew=rzinew+1.0
     end if
c  *****calculate the energy of i in the new config************
     call energy(rxinew,ryinew,rzinew,i,rcut,sigma,vnew,
    +     iflag,eql,foc,vosn,ovrpol,rmin,polymer,neig,bonle,red)
c  *****check for acceptence***********************************
     deltv=vnew-vold
     deltvb=beta*deltv
     delvos=vosn-voso
     if(.not.ovrpol.and.delvos.le.0.0)then
       if(.not.touch) then
             if(deltv.le.0.0) then
             v=v+deltv
             rx(i)=rxinew
             ry(i)=ryinew
             rz(i)=rzinew
             accept=accept+1.0
             else if (exp(-deltvb).gt.rnunf()) then
             v=v+deltv
             rx(i)=rxinew
             ry(i)=ryinew
             rz(i)=rzinew
             accept=accept+1.0
             end if
       end if
     end if
  99 continue
```

```
          acm=acm+1.0
          vn=(v+vlrc)/real(n)
          acv=acv+vn
          acvsq=acvsq+vn*vn
c ********perform periodic operations*****************************
          if (mod(step,iratio).eq.0) then
          ratio=accept/real(n*iratio)
            if (ratio.gt.0.5) then
            drmax=drmax*1.05
        else
            drmax=drmax*0.95
        end if
        accept=0.0
          end if
          if (mod(step,iprint).eq.0) then
          write (*,'(i8,e14.4,e20.6)') int(acm),ratio,vn
      end if
          if (mod(step,isave).eq.0) then
          call writcn (cnfile)
      end if
          call readcn (cnfile)
  100 continue
          write(*,'(//'' end of markov chain ''//)')
      call readcn (cnfile)
c **calculates energy of the configuration and check for overlaps**
      call sumup (rcut,rmin,sigma,ovrlap,vlast,iflag,eql,
     +            foc,vosi,polymer,neig,bonle,red)
          if (abs(vlast-v).gt.control) then
          write(*,'('' problem with energy ,'')')
          write(*,'(''vlast  =  '',e20.6)') vlast
          write(*,'('' v     =  '',e20.6)') v
          end if
          call writcn (cnfile)
          avv=acv/acm
          write(*,'(/'' average ''/)')
          write(*,'(''<v/n>  = '',e20.6)') avv
      write(*,'(''polymer oscil potential= '',e20.6)')vosi
      call readcn (cnfile)
      print *,'i2=   k= ',i2,k
      call poly(iflag,i2,k,bonle,flip,factr,polymer,neig)
      if(flip)go to 16
c  **calculates final energy and check for overlaps**************
      call sumup(rcut,rmin,sigma,ovrlap,vlast,iflag,eql,
     +           foc,vosf,polymer,neig)
      write(*,'(''polymer oscil potentia= '',e20.6)')vosf
  102 continue
   16 voscil=4*vosf/k
      write(*,'(''total oscilat potential= '',e20.6)')vosf
      write(*,'(''average oscil potential= '',e20.6)')voscil
  500    format (t5,e12.4,t19,e12.4,t48,e12.4)
      stop
      end
```

```
      subroutine poly(iflag,i2,k,bonle,flip,factr,polymer,neig)

c ***************************************************************
c **  Forms the cross-linked polymer, and write out polymer  **
c **  file.                                                  **
c **                                                         **
c **  Principal variables:                                   **
c **                                                         **
c **  integer jj(3.4)        up to 3 nearest neighbors for a **
c **                                    given active center **
c **  integer ipoly(n,4)     linear polymer chains           **
c **  integer iflag(n)       bonded or nonbonded status      **
c **  integer i2(1,4)        propagating active centers      **
c **  integer near(n-1)      neighboring status              **
c **  integer m1(1,3),etc    conters for various combinations**
c **  integer m10,m11,etc    of neghboring status            **
c **  real    path(99,4)     distance from active centers to **
c **                                       all other points **
c **  real    path1(3,40)    distance from active centers to **
c **                               up to 3 nearest neighbors **
c **  charact polymer        name of polymer file            **
c ***************************************************************

      common /block1/rx,ry,rz
      common /block2/ipoly
      integer n,jj(3,4),l,m1(1,3),m2(1,2),m3
      integer m10,m11,m12,m13,m14,m20,m21,m22,m23
      parameter (n=100)
      real rx(n),ry(n),rz(n)
      integer ipoly(100,4),iflag(100),i2(1,4),near(99)
      real path(99,4),bonle,rxij,ryij,rzij,path1(3,4)
      logical flip,neig
      character polymer*(*)
      flip=.false.
      do 1 j=1,3
      m1(1,j)=0
    1 continue
      m2(1,1)=0
      m2(1,2)=0
      m10=0
      m11=0
      m12=0
      m13=0
      m14=0
      m20=0
      m21=0
      m22=0
      m23=0
      m31=0
      m32=0
      m33=0
      m34=0
      do 2 j=1,99
      near(j)=0
```

```
2 continue
  do 3 l=1,4
  path1(1,l)=1.2
3 continue
  if(k.eq.1)then
  do 4 l=1,4
  ipoly(k,l)=i2(1,l)
4 continue
  call writpol(polymer)
  return
  else
  do  6 l=1,4
  do  5 i=1,n
  if(i.ne.i2(1,l))then
  rxij=rx(i)-rx(i2(1,l))
  ryij=ry(i)-ry(i2(1,l))
  rzij=rz(i)-rz(i2(1,l))
      if( rxij.gt.0.5) then
      rxij=rxij-1.0
      else if(rxij.lt.-0.5) then
      rxij=rxij+1.0
      end if
      if( ryij.gt.0.5) then
      ryij=ryij-1.0
      else if(ryij.lt.-0.5) then
      ryij=ryij+1.0
      end if
      if( rzij.gt.0.5) then
      rzij=rzij-1.0
      else if(rzij.lt.-0.5) then
      rzij=rzij+1.0
      end if
  path(i,l)=sqrt((rxij)**2+(ryij)**2+(rzij)**2)
  end if
5 continue
6 continue
  do 8 l=1,4
  do 7 i=1,99
  if(iflag(i).eq.2) then
  j=i2(1,l)
  call readpol(polymer,i,j,neig)
    if(neig)then
    near(i)=1
    end if
  end if
  if(i.ne.i2(1,l)) then
      if((iflag(i).ne.0).and.(near(i).eq.0).and.
 +       (path(i,l).lt.path1(1,l)))then
      path1(1,l)=path(i,l)
      jj(1,l)=i
      end if
  end if
7 continue
8 continue
```

```
      do 9 l=1,4
      near(jj(1,l))=1
      path1(2,l)=1.2
    9 continue
      do 11 l=1,4
      do 10 i=1,99
      if((iflag(i).ne.0).and.(near(i).eq.0).and.
     +     (path(i,l).lt.path1(2,l)))then
      path1(2,l)=path(i,l)
      jj(2,l)=i
      end if
   10 continue
   11 continue
      do 12 l=1,4
      near(jj(2,l))=1
      path1(3,l)=1.2
   12 continue
      do 14 l=1,4
      do 13 i=1,99
      if((iflag(i).ne.0).and.(near(i).eq.0).and.
     +    (path(i,l).lt.path1(3,l)))then
      path1(3,l)=path(i,l)
      jj(3,l)=i
      end if
   13 continue
   14 continue
      end if
      do 18 j=1,3
      if(jj(1,j).eq.jj(1,j+1)) then
      m1(1,j)=j
      end if
   18 continue
      do 19 j=1,2
      if (jj(1,j).eq.jj(1,j+2)) then
      m2(1,j)=j
      end if
   19 continue
      if (jj(1,1).eq.jj(1,4)) then
      m3=1
      end if
      if((m1(1,1).eq.0).and.(m1(1,2).eq.0).and.
     +    (m1(1,3).eq.0))then
      m10=1
      end if
      if((m1(1,1).ne.0).and.(m1(1,2).eq.0).and.
     +    (m1(1,3).eq.0))then
      m11=1
      else if((m1(1,1).eq.0).and.(m1(1,2).ne.0).and.
     +         (m1(1,3).eq.0))then
      m12=1
      else if((m1(1,1).eq.0).and.(m1(1,2).eq.0).and.
     +         (m1(1,3).ne.0))then
      m13=1
      else if((m1(1,1).ne.0).and.(m1(1,2).eq.0).and.
```

```
     +         (m1(1,3).ne.0))then
      m14=1
      end if
      if((m2(1,1).eq.0).and.(m2(1,2).eq.0))then
      m20=1
      end if
      if((m2(1,1).ne.0).and.(m2(1,2).eq.0))then
      m21=1
      else if((m2(1,1).eq.0).and.(m2(1,2).ne.0))then
      m22=1
      else if((m2(1,1).ne.0).and.(m2(1,2).ne.0))then
      m23=1
      end if
      if((m1(1,1).ne.0).and.(m1(1,2).ne.0).and.(m21.ne.0))then
      m31=1
      else if((m1(1,3).ne.0).and.(m21.ne.0).and.(m3.ne.0))then
      m32=1
      else if((m1(1,1).ne.0).and.(m22.ne.0).and.(m3.ne.0))then
      m33=1
      else if((m1(1,2).ne.0).and.(m22.ne.0).and.(m3.eq.0))then
      m34=1
      end if
      if ((m10.eq.1).and.(m20.eq.1).and.(m3.eq.0)) then
      do 20 l=1,4
      if (path1(1,l).lt.bonle)then
      ipoly(k,l)=jj(1,l)
      i2(1,l)=jj(1,l)
         if(iflag(jj(1,l)).eq.3) then
         iflag(jj(1,l))=2
         else if(iflag(jj(1,l)).eq.1) then
         iflag(jj(1,l))=0
         else if(iflag(jj(1,l)).eq.2) then
         iflag(jj(1,l))=0
         end if
      end if
   20 continue
      factr=factr*.95
      call writpol(polymer)
      short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
          if (short.gt.bonle)then
          flip=.true.
          end if
      return
      else
      do 21 l=1,4
          if(iflag(jj(1,l)).eq.3)then
          iflag(jj(1,l))=2
          else if(iflag(jj(1,l)).eq.1)then
          iflag(jj(1,l))=0
          else if(iflag(jj(1,l)).eq.2)then
          iflag(jj(1,l))=0
          end if
   21 continue
      end if
```

```
      if ((m3.eq.0).and.(m20.eq.1).and.(m11.eq.1))then
         if (path1(1,1).le.path1(1,2)) then
         jj(1,2)=jj(2,2)
         path1(1,2)=path1(2,2)
          if (iflag(jj(1,2)).eq.3)then
          iflag(jj(1,2))=2
          else if(iflag(jj(1,2)).eq.1)then
          iflag(jj(1,2))=0
          else if(iflag(jj(1,2)).eq.2)then
          iflag(jj(1,2))=0
          end if
         else
         jj(1,1)=jj(2,1)
         path1(1,1)=path1(2,1)
          if(iflag(jj(1,1)).eq.3)then
          iflag(jj(1,1))=2
          else if(iflag(jj(1,1)).eq.1)then
          iflag(jj(1,1))=0
          else if(iflag(jj(1,1)).eq.2)then
          iflag(jj(1,1))=0
          end if
         end if
      do 22 l=1,4
      if(path1(1,1).lt.bonle) then
      ipoly(k,l)=jj(1,1)
      i2(1,l)=jj(1,l)
      end if
   22 continue
      factr=factr*.95
      call writpol(polymer)
      short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
           if(short.gt.bonle) then
           flip=.true.
           end if
      return
      else if((m3.eq.0).and.(m21.eq.1).and.(m10.eq.1)) then
        if(path1(1,1).le.path1(1,3)) then
        jj(1,3)=jj(2,3)
        path1(1,3)=path1(2,3)
          if(iflag(jj(1,3)).eq.3)then
          iflag(jj(1,3))=2
          else if(iflag(jj(1,3)).eq.1)then
          iflag(jj(1,3))=0
          else if(iflag(jj(1,3)).eq.2)then
          iflag(jj(1,3))=0
          end if
        else
        jj(1,1)=jj(2,1)
        path1(1,1)=path1(2,1)
          if(iflag(jj(1,1)).eq.3)then
          iflag(jj(1,1))=2
          else if(iflag(jj(1,1)).eq.1)then
          iflag(jj(1,1))=0
          else if(iflag(jj(1,1)).eq.2)then
```

```
      iflag(jj(1,1))=0
      end if
    end if
  do 23 l=1,4
  if(path1(1,l).lt.bonle)then
  ipoly(k,l)=jj(1,l)
  i2(1,l)=jj(1,l)
  end if
23 continue
  factr=factr*.95
  call writpol(polymer)
  short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
       if(short.gt.bonle) then
       flip=.true.
       end if
  return
  else if((m3.eq.1).and.(m20.eq.1).and.(m10.eq.1))then
    if(path1(1,1).le.path1(1,4)) then
    jj(1,4)=jj(2,4)
    path1(1,4)=path1(2,4)
      if(iflag(jj(1,4)).eq.3)then
      iflag(jj(1,4))=2
      else if(iflag(jj(1,4)).eq.1)then
      iflag(jj(1,4))=0
      else if(iflag(jj(1,4)).eq.2)then
      iflag(jj(1,4))=0
      end if
    else
    jj(1,1)=jj(2,1)
    path1(1,1)=path1(2,1)
      if(iflag(jj(1,1)).eq.3)then
      iflag(jj(1,1))=2
      else if(iflag(jj(1,1)).eq.1)then
      iflag(jj(1,1))=0
      else if(iflag(jj(1,1)).eq.2)then
      iflag(jj(1,1))=0
      end if
    end if
  do 24 l=1,4
  if(path1(1,l).lt.bonle)then
  ipoly(k,l)=jj(1,l)
  i2(1,l)=jj(1,l)
  end if
24 continue
  factr=factr*.95
  call writpol(polymer)
  short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
       if (short.gt.bonle)then
       flip=.true.
       end if
  return
  else if((m3.eq.0).and.(m20.eq.1).and.(m12.eq.1))then
    if(path1(1,2).le.path1(1,3)) then
    jj(1,3)=jj(2,3)
```

```
      path1(1,3)=path1(2,3)
        if(iflag(jj(1,3)).eq.3)then
        iflag(jj(1,3))=2
        else if(iflag(jj(1,3)).eq.1)then
        iflag(jj(1,3))=0
        else if(iflag(jj(1,3)).eq.2)then
        iflag(jj(1,3))=0
        end if
      else
      jj(1,2)=jj(2,2)
      path1(1,2)=path1(2,2)
        if(iflag(jj(1,2)).eq.3)then
        iflag(jj(1,2))=2
        else if(iflag(jj(1,2)).eq.1)then
        iflag(jj(1,2))=0
        else if(iflag(jj(1,2)).eq.2)then
        iflag(jj(1,2))=0
        end if
      end if
    do 25 l=1,4
    if(path1(1,l).lt.bonle)then
    ipoly(k,l)=jj(1,l)
    i2(1,l)=jj(1,l)
    end if
25  continue
    factr=factr*.95
    call writpol(polymer)
    short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
         if(short.gt.bonle)then
         flip=.true.
         end if
    return
    else if((m3.eq.0).and.(m22.eq.1).and.(m10.eq.1))then
      if(path1(1,2).le.path1(1,4)) then
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
        if(iflag(jj(1,4)).eq.3)then
        iflag(jj(1,4))=2
        else if(iflag(jj(1,4)).eq.1)then
        iflag(jj(1,4))=0
        else if(iflag(jj(1,4)).eq.2)then
        iflag(jj(1,4))=0
        end if
      else
      jj(1,2)=jj(2,2)
      path1(1,2)=path1(2,2)
        if(iflag(jj(1,2)).eq.3)then
        iflag(jj(1,2))=2
        else if(iflag(jj(1,2)).eq.1)then
        iflag(jj(1,2))=0
        else if(iflag(jj(1,2)).eq.2)then
        iflag(jj(1,2))=0
        end if
      end if
```

```
      do 26 l=1,4
      if(path1(1,l).lt.bonle)then
      ipoly(k,l)=jj(1,l)
      i2(1,l)=jj(1,l)
      end if
   26 continue
      factr=factr*.95
      call writpol(polymer)
      short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
           if(short.gt.bonle)then
           flip=.true.
           end if
      return
      else if((m3.eq.0).and.(m20.eq.1).and.(m13.eq.1))then
        if(path1(1,3).le.path1(1,4))then
        jj(1,4)=jj(2,4)
        path1(1,4)=path1(2,4)
          if(iflag(jj(1,4)).eq.3)then
          iflag(jj(1,4))=2
          else if(iflag(jj(1,4)).eq.1)then
          iflag(jj(1,4))=0
          else if(iflag(jj(1,4)).eq.2)then
          iflag(jj(1,4))=0
          end if
        else
        jj(1,3)=jj(2,3)
        path1(1,3)=path1(2,3)
          if(iflag(jj(1,3)).eq.3)then
          iflag(jj(1,3))=2
          else if(iflag(jj(1,3)).eq.1)then
          iflag(jj(1,3))=0
          else if(iflag(jj(1,3)).eq.2)then
          iflag(jj(1,3))=0
          end if
        end if
      do 27 l=1,4
      if(path1(1,l).lt.bonle)then
      ipoly(k,l)=jj(1,l)
      i2(1,l)=jj(1,l)
      end if
   27 continue
      factr=factr*.95
      call writpol(polymer)
      short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
           if(short.gt.bonle)then
           flip=.true.
           end if
      return
      else if((m3.eq.0).and.(m20.eq.1).and.(m14.eq.1))then
        if(path1(1,1).le.path1(1,2)) then
        jj(1,2)=jj(2,2)
        path1(1,2)=path1(2,2)
          if(iflag(jj(1,2)).eq.3)then
          iflag(jj(1,2))=2
```

```
      else if(iflag(jj(1,2)).eq.1)then
      iflag(jj(1,2))=0
      else if(iflag(jj(1,2)).eq.2)then
      iflag(jj(1,2))=0
      end if
    else
    jj(1,1)=jj(2,1)
    path1(1,1)=path1(2,1)
      if(iflag(jj(1,1)).eq.3)then
      iflag(jj(1,1))=2
      else if(iflag(jj(1,1)).eq.1)then
      iflag(jj(1,1))=0
      else if(iflag(jj(1,1)).eq.2)then
      iflag(jj(1,1))=0
      end if
    end if
    if(path1(1,3).lt.path1(1,4)) then
    jj(1,4)=jj(2,4)
    path1(1,4)=path1(2,4)
      if(iflag(jj(1,4)).eq.3)then
      iflag(jj(1,4))=2
      else if(iflag(jj(1,4)).eq.1)then
      iflag(jj(1,4))=0
      else if(iflag(jj(1,4)).eq.2)then
      iflag(jj(1,4))=0
       end if
    else
    jj(1,3)=jj(2,3)
    path1(1,3)=path1(2,3)
      if(iflag(jj(1,3)).eq.3)then
      iflag(jj(1,3))=2
      else if(iflag(jj(1,3)).eq.1)then
      iflag(jj(1,3))=0
      else if(iflag(jj(1,3)).eq.2)then
      iflag(jj(1,3))=0
      end if
    end if
  do 28 l=1,4
  if(path1(1,l).lt.bonle) then
  ipoly(k,l)=jj(1,l)
  i2(1,l)=jj(1,l)
  end if
28 continue
  factr=factr*.95
  call writpol(polymer)
  short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
       if(short.gt.bonle) then
       flip=.true.
       end if
  return
  else if((m3.eq.0).and.(m23.eq.1).and.(m10.eq.1))then
    if(path1(1,1).le.path1(1,3)) then
    jj(1,3)=jj(2,3)
    path1(1,3)=path1(2,3)
```

```
      if(iflag(jj(1,3)).eq.3)then
      iflag(jj(1,3))=2
      else if(iflag(jj(1,3)).eq.1)then
      iflag(jj(1,3))=0
      else if(iflag(jj(1,3)).eq.2)then
      iflag(jj(1,3))=0
      end if
    else
    jj(1,1)=jj(2,1)
    path1(1,1)=path1(2,1)
      if(iflag(jj(1,1)).eq.3)then
      iflag(jj(1,1))=2
      else if(iflag(jj(1,1)).eq.1)then
      iflag(jj(1,1))=0
      else if(iflag(jj(1,1)).eq.2)then
      iflag(jj(1,1))=0
      end if
    end if
    if(path1(1,2).le.path1(1,4)) then
    jj(1,4)=jj(2,4)
    path1(1,4)=path1(2,4)
       if(iflag(jj(1,4)).eq.3)then
       iflag(jj(1,4))=2
       else if(iflag(jj(1,4)).eq.1)then
       iflag(jj(1,4))=0
       else if(iflag(jj(1,4)).eq.2)then
       iflag(jj(1,4))=0
       end if
    else
    jj(1,2)=jj(2,2)
    path1(1,2)=path1(2,2)
       if(iflag(jj(1,2)).eq.3)then
       iflag(jj(1,2))=2
       else if(iflag(jj(1,2)).eq.1)then
       iflag(jj(1,2))=0
       else if(iflag(jj(1,2)).eq.2)then
       iflag(jj(1,2))=0
       end if
    end if
  do 29 l=1,4
  if(path1(1,l).lt.bonle) then
  ipoly(k,l)=jj(1,l)
  i2(1,l)=jj(1,l)
  end if
29 continue
  factr=factr*.95
  call writpol(polymer)
  short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
       if(short.gt.bonle) then
       flip=.true.
       end if
  return
  else if((m3.eq.1).and.(m20.eq.1).and.(m12.eq.1))then
    if(path1(1,1).le.path1(1,4)) then
```

```
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
         if(iflag(jj(1,4)).eq.3)then
         iflag(jj(1,4))=2
         else if(iflag(jj(1,4)).eq.1)then
         iflag(jj(1,4))=0
         else if(iflag(jj(1,4)).eq.2)then
         iflag(jj(1,4))=0
         end if
      else
      jj(1,1)=jj(2,1)
      path1(1,1)=path1(2,1)
         if(iflag(jj(1,1)).eq.3)then
         iflag(jj(1,1))=2
         else if(iflag(jj(1,1)).eq.1)then
         iflag(jj(1,1))=0
         else if(iflag(jj(1,1)).eq.2)then
         iflag(jj(1,1))=0
         end if
      end if
      if(path1(1,2).le.path1(1,3))then
      jj(1,3)=jj(2,3)
      path1(1,3)=path1(2,3)
         if(iflag(jj(1,3)).eq.3)then
         iflag(jj(1,3))=2
         else if(iflag(jj(1,3)).eq.1)then
         iflag(jj(1,3))=0
         else if(iflag(jj(1,3)).eq.2)then
         iflag(jj(1,3))=0
         end if
      else
      jj(1,2)=jj(2,2)
      path1(1,2)=path1(2,2)
         if(iflag(jj(1,2)).eq.3)then
         iflag(jj(1,2))=2
         else if(iflag(jj(1,2)).eq.1)then
         iflag(jj(1,2))=0
         else if(iflag(jj(1,2)).eq.2)then
         iflag(jj(1,2))=0
         end if
      end if
    do 30 l=1,4
    if(path1(1,l).lt.bonle) then
    ipoly(k,l)=jj(1,l)
    i2(1,l)=jj(1,l)
    end if
30  continue
    factr=factr*.95
    call writpol(polymer)
    short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
         if(short.gt.bonle) then
         flip=.true.
         end if
    return
```

```
 else if(m31.ne.0)then
   if((path1(1,1).le.path1(1,2)).and.(path1(1,1).
+    le.path1(1,3)).and.(jj(2,2).ne.jj(2,3)))then
   jj(1,2)=jj(2,2)
   path1(1,2)=path1(2,2)
   jj(1,3)=jj(2,3)
   path1(1,3)=path1(2,3)
      if(iflag(jj(1,2)).eq.3)then
      iflag(jj(1,2))=2
      else if(iflag(jj(1,2)).eq.1)then
      iflag(jj(1,2))=0
      else if(iflag(jj(1,2)).eq.2)then
      iflag(jj(1,2))=0
      end if
      if(iflag(jj(1,3)).eq.3)then
      iflag(jj(1,3))=2
      else if(iflag(jj(1,3)).eq.1)then
      iflag(jj(1,3))=0
      else if(iflag(jj(1,3)).eq.2)then
      iflag(jj(1,3))=0
      end if
    end if
    if((path1(1,1).le.path1(1,3)).and.(path1(1,1).le.
+       path1(1,2)).and.(jj(2,2).eq.jj(2,3)))then
    jj(1,3)=jj(2,3)
    path1(1,3)=path1(2,3)
    jj(1,2)=jj(3,2)
    path1(1,2)=path1(3,2)
      if(iflag(jj(1,2)).eq.3)then
      iflag(jj(1,2))=2
      else if(iflag(jj(1,2)).eq.1)then
      iflag(jj(1,2))=0
      else if(iflag(jj(1,2)).eq.2)then
      iflag(jj(1,2))=0
      end if
      if(iflag(jj(1,3)).eq.3)then
      iflag(jj(1,3))=2
      else if(iflag(jj(1,3)).eq.1)then
      iflag(jj(1,3))=0
      else if(iflag(jj(1,3)).eq.2)then
      iflag(jj(1,3))=0
      end if
    end if
    if((path1(1,2).le.path1(1,1)).and.(path1(1,2).le.
+       path1(1,3)).and.(jj(2,1).ne.jj(2,3)))then
    jj(1,1)=jj(2,1)
    path1(1,1)=path1(2,1)
    jj(1,3)=jj(2,3)
    path1(1,3)=path1(2,3)
      if(iflag(jj(1,1)).eq.3)then
      iflag(jj(1,1))=2
      else if(iflag(jj(1,1)).eq.1)then
      iflag(jj(1,1))=0
      else if(iflag(jj(1,1)).eq.2)then
```

```
      iflag(jj(1,1))=0
      end if
      if(iflag(jj(1,3)).eq.3)then
      iflag(jj(1,3))=2
      else if(iflag(jj(1,3)).eq.1)then
      iflag(jj(1,3))=0
      else if(iflag(jj(1,3)).eq.2)then
      iflag(jj(1,3))=0
      end if
    end if
    if((path1(1,2).le.path1(1,1)).and.(path1(1,2).le.
+       path1(1,3)).and.(jj(2,1).eq.jj(2,3)))then
    jj(1,3)=jj(2,3)
    path1(1,3)=path1(2,3)
    jj(1,1)=jj(3,1)
    path1(1,1)=path1(3,1)
      if(iflag(jj(1,1)).eq.3)then
      iflag(jj(1,1))=2
      else if(iflag(jj(1,1)).eq.1)then
      iflag(jj(1,1))=0
      else if(iflag(jj(1,1)).eq.2)then
      iflag(jj(1,1))=0
      end if
      if(iflag(jj(1,3)).eq.3)then
      iflag(jj(1,3))=2
      else if(iflag(jj(1,3)).eq.1)then
      iflag(jj(1,3))=0
      else if(iflag(jj(1,3)).eq.2)then
      iflag(jj(1,3))=0
      end if
    end if
    if((path1(1,3).le.path1(1,1)).and.(path1(1,3).le.
+       path1(1,2)).and.(jj(2,1).ne.jj(2,2)))then
    jj(1,1)=jj(2,1)
    path1(1,1)=path1(2,1)
    jj(1,2)=jj(2,2)
    path1(1,2)=path1(2,2)
       if(iflag(jj(1,1)).eq.3)then
       iflag(jj(1,1))=2
       else if(iflag(jj(1,1)).eq.1)then
       iflag(jj(1,1))=0
       else if(iflag(jj(1,1)).eq.2)then
       iflag(jj(1,1))=0
       end if
       if(iflag(jj(1,2)).eq.3)then
       iflag(jj(1,2))=2
       else if(iflag(jj(1,2)).eq.1)then
       iflag(jj(1,2))=0
       else if(iflag(jj(1,2)).eq.2)then
       iflag(jj(1,2))=0
       end if
     end if
    if((path1(1,3).le.path1(1,2)).and.(path1(1,3).le.
+       path1(1,2)).and.(jj(2,1).eq.jj(2,2)))then
```

```
      jj(1,2)=jj(2,2)
      path1(1,2)=path1(2,2)
      jj(1,1)=jj(3,1)
      path1(1,1)=path1(3,1)
         if(iflag(jj(11,1)).eq.3)then
         iflag(jj(1,1))=2
         else if(iflag(jj(1,1)).eq.1)then
         iflag(jj(1,1))=0
         else if(iflag(jj(1,1)).eq.2)then
         iflag(jj(1,1))=0
         end if
         if(iflag(jj(1,2)).eq.3)then
         iflag(jj(1,2))=2
         else if(iflag(jj(1,2)).eq.1)then
         iflag(jj(1,2))=0
         else if(iflag(jj(1,2)).eq.2)then
         iflag(jj(1,2))=0
         end if
       end if
   do 31 l=1,4
   if(path1(1,l).lt.bonle)then
   ipoly(k,l)=jj(1,l)
   i2(1,l)=jj(1,l)
   end if
31 continue
   factr=factr*.95
   call writpol(polymer)
   short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
        if(short.gt.bonle) then
        flip=.true.
        end if
   return
   else if(m32.ne.0) then
       if((path1(1,1).le.path1(1,3)).and.(path1(1,1).
  +       le.path1(1,4)).and.(jj(2,3).ne.path1(2,4)))then
       jj(1,3)=jj(2,3)
       path1(1,3)=path1(2,3)
       jj(1,4)=jj(2,4)
       path1(1,4)=path1(2,4)
         if(iflag(jj(1,3)).eq.3)then
         iflag(jj(1,3))=2
         else if(iflag(jj(1,3)).eq.1)then
         iflag(jj(1,3))=0
         else if(iflag(jj(1,3)).eq.2)then
         iflag(jj(1,3))=0
         end if
         if(iflag(jj(1,4)).eq.3)then
         iflag(jj(1,4))=2
         else if(iflag(jj(1,4)).eq.1)then
         iflag(jj(1,4))=0
         else if(iflag(jj(1,4)).eq.2)then
         iflag(jj(1,4))=0
         end if
       end if
```

```
      if((path1(1,1).le.path1(1,4)).and.(path1(1,1).le.
     +     path1(1,3)).and.(jj(2,3).eq.jj(2,4)))then
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
      jj(1,3)=jj(3,3)
      path1(1,3)=path1(3,3)
        if(iflag(jj(1,4)).eq.3)then
        iflag(jj(1,4))=2
        else if(iflag(jj(1,4)).eq.1)then
        iflag(jj(1,4))=0
        else if(iflag(jj(1,4)).eq.2)then
        iflag(jj(1,4))=0
        end if
        if(iflag(jj(1,3)).eq.3)then
        iflag(jj(1,3))=2
        else if(iflag(jj(1,3)).eq.1)then
        iflag(jj(1,3))=0
        else if(iflag(jj(1,3)).eq.2)then
        iflag(jj(1,3))=0
        end if
      end if
      if((path1(1,3).le.path1(1,1)).and.(path1(1,3).le.
     +     path1(1,4)).and.(jj(2,1).ne.jj(2,4)))then
      jj(1,1)=jj(2,1)
      path1(1,1)=path1(2,1)
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
        if(iflag(jj(1,1)).eq.3)then
        iflag(jj(1,1))=2
        else if(iflag(jj(1,1)).eq.1)then
        iflag(jj(1,1))=0
        else if(iflag(jj(1,1)).eq.2)then
        iflag(jj(1,1))=0
        end if
        if(iflag(jj(1,4)).eq.3)then
        iflag(jj(1,4))=2
        else if(iflag(jj(1,4)).eq.1)then
        iflag(jj(1,4))=0
        else if(iflag(jj(1,4)).eq.2)then
        iflag(jj(1,4))=0
        end if
      end if
      if((path1(1,3).le.path1(1,4)).and.(path1(1,3).le.
     +     path1(1,1)).and.(jj(2,1).eq.jj(2,4)))then
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
      jj(1,1)=jj(3,1)
      path1(1,1)=path1(3,1)
        if(iflag(jj(1,1)).eq.3)then
        iflag(jj(1,1))=2
        else if(iflag(jj(1,1)).eq.1)then
        iflag(jj(1,1))=0
        else if(iflag(jj(1,1)).eq.2)then
        iflag(jj(1,1))=0
```

```
        end if
        if(iflag(jj(1,4)).eq.3)then
        iflag(jj(1,4))=2
        else if(iflag(jj(1,4)).eq.1)then
        iflag(jj(1,4))=0
        else if(iflag(jj(1,4)).eq.2)then
        iflag(jj(1,4))=0
        end if
      end if
      if((path1(1,4).le.path1(1,1)).and.(path1(1,4).le.
     +     path1(1,3)).and.(jj(2,1).ne.jj(2,3)))then
      jj(1,1)=jj(2,1)
      path1(1,1)=path1(2,1)
      jj(1,3)=jj(2,3)
      path1(1,3)=path1(2,3)
        if(iflag(jj(1,1)).eq.3)then
        iflag(jj(1,1))=2
        else if(iflag(jj(1,1)).eq.1)then
        iflag(jj(1,1))=0
        else if(iflag(jj(1,1)).eq.2)then
        iflag(jj(1,1))=0
        end if
        if(iflag(jj(1,3)).eq.3)then
        iflag(jj(1,3))=2
        else if(iflag(jj(1,3)).eq.1)then
        iflag(jj(1,3))=0
        else if(iflag(jj(1,3)).eq.2)then
        iflag(jj(1,3))=0
        end if
      end if
      if((path1(1,4).le.path1(1,3)).and.(path1(1,4).le.
     +     path1(1,1)).and.(jj(2,1).eq.jj(2,3)))then
      jj(1,3)=jj(2,3)
      path1(1,3)=path1(2,3)
      jj(1,1)=jj(3,1)
      path1(1,1)=path1(3,1)
        if(iflag(jj(1,1)).eq.3)then
        iflag(jj(1,1))=2
        else if(iflag(jj(1,1)).eq.1)then
        iflag(jj(1,1))=0
        else if(iflag(jj(1,1)).eq.2)then
        iflag(jj(1,1))=0
        end if
        if(iflag(jj(1,3)).eq.3)then
        iflag(jj(1,3))=2
        else if(iflag(jj(1,3)).eq.1)then
        iflag(jj(1,3))=0
        else if(iflag(jj(1,3)).eq.2)then
        iflag(jj(1,3))=0
        end if
      end if
 do 32 l=1,4
 if(path1(1,l).lt.bonle)then
 ipoly(k,l)=jj(1,l)
```

```
      i2(1,1)=jj(1,1)
      end if
   32 continue
      factr=factr*.95
      call writpol(polymer)
      short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
           if(short.gt.bonle) then
           flip=.true.
           end if
      return
      else if(m33.ne.0) then
          if((path1(1,1).le.path1(1,2)).and.(path1(1,1).
     +      le.path1(1,4)).and.(jj(2,2).ne.jj(2,4)))then
          jj(1,2)=jj(2,2)
          path1(1,2)=path1(2,2)
          jj(1,4)=jj(2,4)
          path1(1,4)=path1(2,4)
            if(iflag(jj(1,2)).eq.3)then
            iflag(jj(1,2))=2
            else if(iflag(jj(1,2)).eq.1)then
            iflag(jj(1,2))=0
            else if(iflag(jj(1,2)).eq.2)then
            iflag(jj(1,2))=0
            end if
            if(iflag(jj(1,4)).eq.3)then
            iflag(jj(1,4))=2
            else if(iflag(jj(1,4)).eq.1)then
            iflag(jj(1,4))=0
            else if(iflag(jj(1,4)).eq.2)then
            iflag(jj(1,4))=0
            end if
          end if
          if((path1(1,1).le.path1(1,4)).and.(path1(1,1).le.
     +        path1(1,2)).and.(jj(2,4).eq.jj(2,2)))then
          jj(1,4)=jj(2,4)
          path1(1,4)=path1(2,4)
          jj(1,2)=jj(3,2)
          path1(1,2)=path1(3,2)
            if(iflag(jj(1,2)).eq.3)then
            iflag(jj(1,2))=2
            else if(iflag(jj(1,2)).eq.1)then
            iflag(jj(1,2))=0
            else if(iflag(jj(1,2)).eq.2)then
            iflag(jj(1,2))=0
            end if
            if(iflag(jj(1,4)).eq.3)then
            iflag(jj(1,4))=2
            else if(iflag(jj(1,4)).eq.1)then
            iflag(jj(1,4))=0
            else if(iflag(jj(1,4)).eq.2)then
            iflag(jj(1,4))=0
            end if
          end if
          if((path1(1,2).le.path1(1,1)).and.(path1(1,2).le.
```

```
+         path1(1,4)).and.(jj(2,1).ne.jj(2,4)))then
      jj(1,1)=jj(2,1)
      path1(1,1)=path1(2,1)
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
        if(iflag(jj(1,1)).eq.3)then
        iflag(jj(1,1))=2
        else if(iflag(jj(1,1)).eq.1)then
        iflag(jj(1,1))=0
        else if(iflag(jj(1,1)).eq.2)then
        iflag(jj(1,1))=0
        end if
        if(iflag(jj(1,4)).eq.3)then
        iflag(jj(1,4))=2
        else if(iflag(jj(1,4)).eq.1)then
        iflag(jj(1,4))=0
        else if(iflag(jj(1,4)).eq.2)then
        iflag(jj(1,4))=0
        end if
      end if
      if((path1(1,2).le.path1(1,4)).and.(path1(1,2).le.
+         path1(1,1)).and.(jj(2,1).eq.jj(2,4)))then
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
      jj(1,1)=jj(3,1)
      path1(1,1)=path1(3,1)
        if(iflag(jj(1,1)).eq.3)then
        iflag(jj(1,1))=2
        else if(iflag(jj(1,1)).eq.1)then
        iflag(jj(1,1))=0
        else if(iflag(jj(1,1)).eq.2)then
        iflag(jj(1,1))=0
        end if
        if(iflag(jj(1,4)).eq.3)then
        iflag(jj(1,4))=2
        else if(iflag(jj(1,4)).eq.1)then
        iflag(jj(1,4))=0
        else if(iflag(jj(1,4)).eq.2)then
        iflag(jj(1,4))=0
        end if
      end if
      if((path1(1,4).le.path1(1,1)).and.(path1(1,4).le.
+         path1(1,2)).and.(jj(2,1).ne.jj(2,2)))then
      jj(1,1)=jj(2,1)
      path1(1,1)=path1(2,1)
      jj(1,2)=jj(2,2)
      path1(1,2)=path1(2,2)
        if(iflag(jj(1,1)).eq.3)then
        iflag(jj(1,1))=2
        else if(iflag(jj(1,1)).eq.1)then
        iflag(jj(1,1))=0
        else if(iflag(jj(1,1)).eq.2)then
        iflag(jj(1,1))=0
        end if
```

```
        if(iflag(jj(1,2)).eq.3)then
        iflag(jj(1,2))=2
        else if(iflag(jj(1,2)).eq.1)then
        iflag(jj(1,2))=0
        else if(iflag(jj(1,2)).eq.2)then
        iflag(jj(1,2))=0
        end if
      end if
      if((path1(1,4).le.path1(1,2)).and.(path1(1,4).le.
     +     path1(1,1)).and.(jj(2,1).eq.jj(2,2)))then
      jj(1,2)=jj(2,2)
      path1(1,2)=path1(2,2)
      jj(1,1)=jj(3,1)
      path1(1,1)=path1(3,1)
        if(iflag(jj(1,1)).eq.3)then
        iflag(jj(1,1))=2
        else if(iflag(jj(1,1)).eq.1)then
        iflag(jj(1,1))=0
        else if(iflag(jj(1,1)).eq.2)then
        iflag(jj(1,1))=0
        end if
        if(iflag(jj(1,2)).eq.3)then
        iflag(jj(1,2))=2
        else if(iflag(jj(1,2)).eq.1)then
        iflag(jj(1,2))=0
        else if(iflag(jj(1,2)).eq.2)then
        iflag(jj(1,2))=0
        end if
      end if
   do 33 l=1,4
   if(path1(1,l).lt.bonle)then
   ipoly(k,l)=jj(1,l)
   i2(1,l)=jj(1,l)
   end if
33 continue
   factr=factr*.95
   call writpol(polymer)
   short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
        if(short.gt.bonle) then
        flip=.true.
        end if
   return
   else if(m34.ne.0) then
      if((path1(1,2).le.path1(1,3)).and.(path1(1,2).
     +     le.path1(1,4)).and.(jj(2,3).eq.jj(2,4)))then
      jj(1,3)=jj(2,3)
      path1(1,3)=path1(2,3)
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
        if(iflag(jj(1,3)).eq.3)then
        iflag(jj(1,3))=2
        else if(iflag(jj(1,3)).eq.1)then
        iflag(jj(1,3))=0
        else if(iflag(jj(1,3)).eq.2)then
```

```
        iflag(jj(1,3))=0
        end if
        if(iflag(jj(1,4)).eq.3)then
        iflag(jj(1,4))=2
        else if(iflag(jj(1,4)).eq.1)then
        iflag(jj(1,4))=0
        else if(iflag(jj(1,4)).eq.2)then
        iflag(jj(1,4))=0
        end if
      end if
      if((path1(1,2).le.path1(1,4)).and.(path1(1,2).le.
     +     path1(1,3)).and.(jj(2,4).eq.jj(2,3)))then
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
      jj(1,3)=jj(3,3)
      path1(1,3)=path1(3,3)
        if(iflag(jj(1,3)).eq.3)then
        iflag(jj(1,3))=2
        else if(iflag(jj(1,3)).eq.1)then
        iflag(jj(1,3))=0
        else if(iflag(jj(1,3)).eq.2)then
        iflag(jj(1,3))=0
        end if
        if(iflag(jj(1,4)).eq.3)then
        iflag(jj(1,4))=2
        else if(iflag(jj(1,4)).eq.1)then
        iflag(jj(1,4))=0
        else if(iflag(jj(1,4)).eq.2)then
        iflag(jj(1,4))=0
        end if
      end if
      if((path1(1,3).le.path1(1,2)).and.(path1(1,3).le.
     +     path1(1,4)).and.(jj(2,2).ne.jj(2,4)))then
      jj(1,2)=jj(2,2)
      path1(1,2)=path1(2,2)
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
        if(iflag(jj(1,2)).eq.3)then
        iflag(jj(1,2))=2
        else if(iflag(jj(1,2)).eq.1)then
        iflag(jj(1,2))=0
        else if(iflag(jj(1,2)).eq.2)then
        iflag(jj(1,2))=0
        end if
        if(iflag(jj(1,4)).eq.3)then
        iflag(jj(1,4))=2
        else if(iflag(jj(1,4)).eq.1)then
        iflag(jj(1,4))=0
        else if(iflag(jj(1,4)).eq.2)then
        iflag(jj(1,4))=0
        end if
      end if
      if((path1(1,3).le.path1(1,4)).and.(path1(1,3).le.
     +     path1(1,2)).and.(jj(2,4).eq.jj(2,2)))then
```

```
      jj(1,4)=jj(2,4)
      path1(1,4)=path1(2,4)
      jj(1,2)=jj(3,2)
      path1(1,2)=path1(3,2)
        if(iflag(jj(1,2)).eq.3)then
        iflag(jj(1,2))=2
        else if(iflag(jj(1,2)).eq.1)then
        iflag(jj(1,2))=0
        else if(iflag(jj(1,2)).eq.2)then
        iflag(jj(1,2))=0
        end if
        if(iflag(jj(1,4)).eq.3)then
        iflag(jj(1,4))=2
        else if(iflag(jj(1,4)).eq.1)then
        iflag(jj(1,4))=0
        else if(iflag(jj(1,4)).eq.2)then
        iflag(jj(1,4))=0
        end if
      end if
      if((path1(1,4).le.path1(1,2)).and.(path1(1,4).le.
     +     path1(1,3)).and.(jj(2,2).ne.jj(2,3)))then
      jj(1,2)=jj(2,2)
      path1(1,2)=path1(2,2)
      jj(1,4)=jj(2,3)
      path1(1,4)=path1(2,3)
        if(iflag(jj(1,2)).eq.3)then
        iflag(jj(1,2))=2
        else if(iflag(jj(1,2)).eq.1)then
        iflag(jj(1,2))=0
        else if(iflag(jj(1,2)).eq.2)then
        iflag(jj(1,2))=0
        end if
        if(iflag(jj(1,3)).eq.3)then
        iflag(jj(1,3))=2
        else if(iflag(jj(1,3)).eq.1)then
        iflag(jj(1,3))=0
        else if(iflag(jj(1,3)).eq.2)then
        iflag(jj(1,3))=0
        end if
      end if
      if((path1(1,4).le.path1(1,3)).and.(path1(1,4).le.
     +     path1(1,2)).and.(jj(2,2).eq.jj(2,3)))then
      jj(1,3)=jj(2,3)
      path1(1,3)=path1(2,3)
      jj(1,2)=jj(3,2)
      path1(1,2)=path1(3,2)
        if(iflag(jj(1,2)).eq.3)then
        iflag(jj(1,2))=2
        else if(iflag(jj(1,2)).eq.1)then
        iflag(jj(1,2))=0
        else if(iflag(jj(1,2)).eq.2)then
        iflag(jj(1,2))=0
        end if
        if(iflag(jj(1,3)).eq.3)then
```

```
            iflag(jj(1,3))=2
            else if(iflag(jj(1,3)).eq.1)then
            iflag(jj(1,3))=0
            else if(iflag(jj(1,3)).eq.2)then
            iflag(jj(1,3))=0
            end if
          end if
      do 34 l=1,4
      if(path1(1,l).lt.bonle)then
      ipoly(k,l)=jj(1,l)
      i2(1,l)=jj(1,l)
      end if
   34 continue
      factr=factr*.95
      call writpol(polymer)
      short=amin1(path1(1,1),path1(1,2),path1(1,3),path1(1,4))
           if(short.gt.bonle) then
           flip=.true.
           end if
      return
      end if
          return
          end
c*********************************************************************

      subroutine sumup(rcut,rmin,sigma,ovrlap,v,iflag,eql,foc,
     +                 vos,polymer,neig,red)

      common /block1/rx,ry,rz
      common /block2/ipoly
c*********************************************************************
c  **                                                               **
c  ** Calculates the total potential energy                         **
c  ** For bonded neighbors calculates oscilator energy and for      **
c  ** others calculates LJ energy.                                  **
c  **                                                               **
c  ** Principal variables:                                          **
c  **                                                               **
c  ** integer n              the number of atoms                    **
c  ** integer iflag(n)       bonded status                          **
c  ** integer ipoly(n,4)     linear polymer chains                  **
c  ** real    rx,ry,rz       the positions of the atoms             **
c  ** real    v              the LJ potential energy                **
c  ** real    vos            the oscilator potential energy         **
c  ** real    rmin           LJ overlaping distance                 **
c  ** real    rmin1          oscilatoroverlaping distance           **
c  ** logical overlap        true for substantial atom overlap      **
c  ** logical neig           true if i and j are neighbors in       **
c  **                        any linear chain                       **
c  **                                                               **
c  ** Returns the total potential energy at the begining and        **
c  ** at the end of the run.Check for the overlaps in oscilator     **
c  ** potential and the LJ potential separetely.                     **
c  *******************************************************************
```

```
      integer n
      parameter (n=100)
      real rx(n),ry(n),rz(n)
      real sigma,rmin,rcut,v,foc,vos,red,rmin1
      logical ovrlap,neig
      real rcutsq,rminsq,sigsq,rxij,ryij,rzij,bonle,bon11
      real rxi,ryi,rzi,vij,sr2,sr6,rijsq,eq1,rmisq1
      integer i,j,iflag(100),ipoly(100,4)
c     ****************************************************************
      character polymer*(*)
      overlap=.false.
      rcutsq=rcut*rcut
      rminsq=rmin*rmin
      rmin1=rmin*red
      rmisq1=rmin1*rmin1
      bon11=bonle*1.3
      sigsq=sigma*sigma
      v=0.0
      vos=0.0
      do 100 i=1,n-1
      rxi=rx(i)
      ryi=ry(i)
      rzi=rz(i)
      do 99 j=i+1,n
      rxij=rxi-rx(j)
      ryij=ryi-ry(j)
      rzij=rzi-rz(j)
c     ******* minimum image pair separations************************
      if( rxij.gt.0.5) then
          rxij=rxij-1.0
          else if(rxij.lt.-0.5) then
          rxij=rxij+1.0
          end if
          if( ryij.gt.0.5) then
          ryij=ryij-1.0
          else if(ryij.lt.-0.5) then
          ryij=ryij+1.0
          end if
          if( rzij.gt.0.5) then
      rzij=rzij-1.0
          else if(rzij.lt.-0.5) then
          rzij=rzij+1.0
          end if
      rijsq=rxij*rxij+ryij*ryij+rzij*rzij
          if (rijsq.lt.rminsq) then
          ovrlap=.true.
          return
      if((iflag(i).eq.0.or.iflag(i).eq.2).and.
     +         (iflag(j).eq.0.or.iflag(j).eq.2))then
        if (rijsq.lt.bon11) then
        call readpol(polymer,i,j,neig)
             if(neig)then
                if (rijsq.lt.rmisq1) then
                ovrlap=.true.
```

```
            return
            else
            rij=sqrt(rijsq)
            vos=vos+0.5*foc*(rij-eql)**2
            end if
         else if((rijsq.lt.rminsq) then
         ovrlap=.true.
         return
         end if
       end if
     else if (rijsq.lt.rcutsq) then
          if((rijsq.lt.rminsq) then
         ovrlap=.true.
         return
         else
         sr2=sigsq/rijsq
         sr6=sr2*sr2*sr2
         vij=sr6*(sr6-1.0)
         v=v+vij
         end if
     end if
  99 continue
 100 continue
         v=4.0*v
         return
         end

c*****************************************************************

      subroutine energy(rxi,ryi,rzi,i,rcut,sigma,v,iflag,eql,
     +                  foc,vos,ovrpol,rmin,polymer,neig,red)

      common /block1/rx,ry,rz
      common /block2/ipoly

c*****************************************************************
c  ** This subroutine is used to calculate the change of     *
c  ** energy during a trial move of atom i.it is caled before*
c  ** and after the random displacement of i. For bonded    **
c  ** neighbors displacements are accected towrds the equli.**
c  **                                                       **
c  ** Principal variables:                                  **
c  **                                                       **
c  ** integer   n          number of monomers               **
c  ** integer   iflag(n)   bonded status                    **
c  ** integer   ipoly(n,4) linear polymer chains            **
c  ** logical   ovrpol     true if 2 neighbors in a linear  **
c  **                      chain overlaps or they exceed the**
c  **                      maximum bond length              **
c  ** logical   touch      true for all other overlaps      **
c  ** logical   neig       true if i and j are neighbors in **
c  **                                    a linear chain **
c  ** character polymer    name of polymer file             **
c*****************************************************************
```

```
      integer n,i
      parameter (n=100)
      real rx(n),ry(n),rz(n)
      real rcut,sigma,rxi,ryi,rzi,v,rmin
      real rcutsq,sigsq,sr2,sr6,eql
      real rxij,ryij,rzij,rijsq,vij
      integer j,iflag(100),ipoly(100,4)
      character polymer*(*)
      logical ovrpol,neig,touch

c  ***********************************************************
      ovrpol=.false.
      touch =.false.
      rcutsq=rcut*rcut
      rminsq=rmin*rmin
      sigsq =sigma*sigma
      v=0.0
      vos=0.0
c  *****loop over all molecules except i*********************

      do 100 j=1,n
          if(i.ne.j)then
          rxij=rxi-rx(j)
          ryij=ryi-ry(j)
          rzij=rzi-rz(j)
          if (rxij.gt.0.5) then
          rxij=rxij-1.0
      else if (rxij.lt.-0.5) then
          rxij=rxij+1.0
      end if
      if (ryij.gt.0.5) then
          ryij=ryij-1.0
      else if (ryij.lt.-0.5) then
          ryij=ryij+1.0
      end if
          if (rzij.gt.0.5) then
          rzij=rzij-1.0
          else if (rzij.lt.-0.5) then
          rzij=rzij+1.0
          end if
      rijsq=(rxij*rxij+ryij*ryij+rzij*rzij)
      if((iflag(i).eq.0.or.iflag(i).eq.2).and.
     +    (iflag(j).eq.0. or.iflag(j).eq.2))then
         if (rijsq.lt.bonl1) then
         call readpol(polymer,i,j,neig)
            if(neig)then
               if((rijsq.lt.rmisq1).or.rijsq.gt.bonle)) then
               ovrpol=.true.
               return
               else
               rij=sqrt(rijsq)
               vos=vos+0.5*foc*(rij-eql)**2
               end if
          else if((rijsq.lt.rminsq) then
```

```
        ovrpol=.true.
        return
        end if
      end if
 else if (rijsq.lt.rcutsq) then
        if(rijsq.lt.rminsq) then
        touch=.true.
        return
        else
        sr2=sigsq/rijsq
        sr6=sr2*sr2*sr2
        vij=sr6*(sr6-1.0)
        v=v+vij
        end if
 end if
100 continue
    v=4.0*v
    return
    end

c  ****************************************************************

    subroutine readcn (cnfile )

        common /block1/rx,ry,rz

c  ****************************************************************
c  ** Subroutine to read cordintes from unit 10                   **
c  ** Reads cordinates for new MOnte Carlo cycle                  **
c  **                                                            **
c  ** Principal variables:                                       **
c  **                                                            **
c  ** integer     n           number of monomers                 **
c  ** real        rx(n),etc   cordinates of mpnomer units        **
c  ****************************************************************

        integer n
    parameter (n=100)
        real rx(n),ry(n),rz(n)
        character cnfile*(*)
        integer cnunit
    parameter (cnunit=10)
        open(unit=cnunit,file='cnfile',status='old',
   +     form='formatted')
        rewind(unit=cnunit)
        do 30 i=1,n
        read (unit= cnunit,fmt=500,end=31) rx(i), ry(i),rz(i)
 30 continue
 31 write(*,'(''end of file read'')')
    close (unit=cnunit)
500 format(t5,e12.4,t19,e12.4,t48,e12.4)
        return
        end
```

```
c ****************************************************************
      subroutine writcn (cnfile)

      common /block1/rx,ry,rz

c*****************************************************************
c  ** Subroutine to write cordinate file after each MC cycle  **
c  **                                                         **
c  ** Principal variables:                                    **
c  **                                                         **
c  ** integer     n             number of monomer units       **
c  ** real        rx(n),etc     cordinates if monomer units   **
c  *************************************************************

      integer n
          parameter (n=100)
          character cnfile*(*)
          real rx(n),ry(n),rz(n)
          integer  cnunit
      parameter (cnunit=10)
          open (unit=cnunit,file='cnfile',form='formatted')
          do 40 i=1,n
          write (cnunit,500) rx(i),ry(i),rz(i)
   40 continue
          endfile (unit=cnunit)
          close (unit=cnunit)
  500 format(t5,e12.4,t19,e12.4,t48,e12.4)
      return
      end

c  ****************************************************************

      subroutine readpol(polymer,i,j,neig)

      common /block2/ipoly

c  ****************************************************************
c  ** Subruotine to read polymer file and identify neighbors  **
c  **                                                         **
c  ** Principal variables:                                    **
c  **                                                         **
c  ** integer   ipoly(n,4)   polymer chains                   **
c  ** logical   neig         true if i and j are neighbors in **
c  **                                  the same linear chain **
c  ****************************************************************

      character polymer*(*)
      logical neig
      integer i,j,ipoly(100,4),j3,j4,jcoun,j5,j6,j7,j8,j9,j10
      neig=.false.
      open(12,file='polymer',status='unknown',form='formatted')
      rewind (unit=12)
      read(12,fmt=502,end=25)((ipoly(jcoun,lco),lco=1,4),
     +      jcoun=1,100)
```

```
      do 17 jcoun=1,100
      do 16 lco=1,4
      if(ipoly(jcoun,lco).eq.i) go to 18
   16 continue
   17 continue
   18 j3=(jcoun-1)
      j4=lco
      j7=(jcoun+1)
      do 20 jcoun=1,100
      do 19 loc=1,4
      if(ipoly(jcoun,lco).eq.j) go to 21
   19 continue
   20 continue
   21 j5=jcoun
      j6=lco
      if((j3.eq.j5.and.j4.eq.j6).or.(j7.eq.j5.and.j4.eq.j6)then
      neig=.true.
      go to 25
      end if
      do 23 jcoun=100,0,-1
      do 22 loc=1,4
      if(ipoly(jcoun,lco).eq.j) go to 24
   22 continue
   23 continue
   24 j8=jcoun-1
      j9=lco
      j10=jcoun+1
      if((j8.eq.j5.and.j9.eq.j6).or.(j10.eq.j5.and.j9.eq.j6)then
      neig=.true.
      end if
   25 close(unit=12)
  502 format(100(4i10/))
      return
      end

c  ****************************************************************

      subroutine writpol(polymer)

      common /block2/ipoly

c  ****************************************************************
c  ** subroutine to write polymer file at end of each MC cycle**
c  **                                                            **
c  ** Principal variables:                                       **
c  **                                                            **
c  ** integer  ipoly(n,4)  polymer chains                        **
c  ****************************************************************

      integer ipoly(100,4),jcoun
      character polymer*(*)
      open(12,file='polymer',status='unknown',form='formatted')
      write(12,fmt=501)((ipoly(jcoun,lco),lco=1,4),jcoun=1,100)
      endfile(unit=12)
```

```
      close(unit=12)
  501 format(100(4i10/))
      return
      end
 ***********************************************************************
```

APPENDIX C

This appendix contains some programs need to analyze the structure of the reaction system at different stages of the growth.

1. Program 'CONNECT' will read 'polymer' file and write the status of all particles (bonded or non-bonded and if bonded particle numbers) to the 'link' file.

2. Program 'IMAGE' will read 'cnfile' file and 'polymer' file and write the information (to file 'disc') of which bonded pairs are image connection.

3. Program 'CONNECT2' will read 'link' file and 'disc' file and write the information of bonded neighbors which are not image connections to the 'linkf' file.

```
program connect
common /block2/ipoly
common /block3/jpoly
integer n,m
parameter (n=934)
parameter (m=200)
integer jcoun,lco,j3,j4,ipoly(m,4),j1,k1,jpoly(n,5)
integer iflag(n),j5,j6,j7,j8,j9,j10
character polymer*10,link*6
call readpol(polymer)
do 17 lco=1,4
do 16 jcoun=1,200
i=ipoly(jcoun,lco)
if(iflag(i).eq.0)then
j3=(jcoun+1)
j4=(jcoun-1)
j1=ipoly(j3,lco)
k1=ipoly(j4,lco)
if(j1.lt.k1) then
j=j1
k=k1
else
j=k1
k=j1
end if
jpoly(i,1)=i
jpoly(i,2)=j
jpoly(i,3)=k
call writlink(link)
iflag(i)=1
else
j5=(jcoun+1)
j6=(jcoun-1)
j1=ipoly(j5,lco)
k1=ipoly(j6,lco)
if(j1.lt.k1)then
j=j1
k=k1
else
j=k1
k=j1
end if
jpoly(i,4)=j
jpoly(i,5)=k
if(jpoly(i,3).gt.jpoly(i,4))then
j7=jpoly(i,3)
jpoly(i,3)=jpoly(i,4)
jpoly(i,4)=j7
end if
if(jpoly(i,2).gt.jpoly(i,3))then
j8=jpoly(i,2)
jpoly(i,2)=jpoly(i,3)
jpoly(i,3)=j8
end if
```

```
      if(jpoly(i,4).gt.jpoly(i,5))then
      j9=jpoly(i,4)
      jpoly(i,4)=jpoly(i,5)
      jpoly(i,5)=j9
      end if
      if(jpoly(i,3).gt.jpoly(i,4))then
      j10=jpoly(i,3)
      jpoly(i,3)=jpoly(i,4)
      jpoly(i,4)=j10
      end if
      call writlink(link)
      end if
16    continue
17    continue
      stop
501   format(934(5i6/))
502   format(200(4i10/))
      end
      subroutine readpol(polymer)
      common /block2/ipoly
      integer m
      parameter (m=200)
      integer jcoun,lco,ipoly(200,4)
      open(12,file='polymer',status='unknown',form='formatted')
      rewind(unit=12)
      read(12,fmt=502,end=22)((ipoly(jcoun,lco),lco=1,4),
     +                        jcoun=1,200)
22    close(unit=12)
502   format(200(4i10/))
      return
      end
      subroutine writlink(link)
      common /block3/jpoly
      integer jpoly(934,5),jco,lco
      open(14,file='link',status='unknown',form='formatted')
      write(14,fmt=501)((jpoly(jco,lco),lco=1,5),jco=1,934)
      endfile(unit=14)
      close(unit=14)
501   format(934(5i6/))
      return
      end
```

```
      program image
      common /block1/rx,ry,rz
      common /block2/ipoly
      common /block3/dis
      integer n,m
      parameter (n=934)
      parameter (m=200)
      integer ipoly(m,4),dis(m,2)
      real rx(n),ry(n),rz(n)
      real path(1,4),rxij,ryij,rzij
      character cnfile*10,polymer*10,disc*10
      call readcn(cnfile)
      call readpol(polymer)
      j=0
      do 2 i=1,m
      do 1 l=1,4
      if((ipoly(i,l).ne.0).and.(ipoly(i+1,l).ne.0)) then
      rxij=rx(ipoly(i+1,l))-rx(ipoly(i,l))
      ryij=ry(ipoly(i+1,l))-ry(ipoly(i,l))
      rzij=rz(ipoly(i+1,l))-rz(ipoly(i,l))
      path(1,l)=sqrt((rxij)**2+(ryij)**2+(rzij)**2)
      end if
c      print*,'path(1,l)= ',path(1,l)
      if (path(1,l).gt.1.0) then
      j=j+1
      dis(j,1)=ipoly(i,l)
      dis(j,2)=ipoly(i+1,l)
      print *,'i,j = ',ipoly(i,l),ipoly(i+1,l)
      end if
1     continue
2     continue
      call wridisc(disc)
      stop
      end
      subroutine readpol(polymer)
      common /block2/ipoly
      integer n
      parameter (n=934)
      parameter (m=200)
      character polymer*(*)
      integer i,l,ipoly(m,4)
      open(12,file='polymer',status='unknown',form='formatted')
      rewind (unit=12)
      read(12,fmt=502,end=22)((ipoly(i,l),l=1,4),i=1,200)
22     close(unit=12)
502    format(200(4i10/))
      return
      end
      subroutine readcn (cnfile )
C     ** SUBROUTINE TO READ IN THE CONFIGURATION FROM UNIT 10          **
      common /block1/rx,ry,rz
      integer n
      parameter (n=934)
      real rx(n),ry(n),rz(n)
```

```
      character cnfile*(*)
      integer cnunit
      parameter (cnunit=10)
      open(unit=cnunit,file='cnfile',status='old',form='formatted')
      rewind(unit=cnunit)
      do 30 i=1,n
      read (unit= cnunit,fmt=500,end=31) rx(i), ry(i),rz(i)
30    continue
31    write(*,'(''end of file read'')')
      close (unit=cnunit)
500    format(t5,e12.4,t19,e12.4,t48,e12.4)
      return
      end
      subroutine wridisc(disc)
      common /block3/dis
      integer m
      parameter (m=200)
      integer dis(m,2)
      character disc*(*)
      open(16,file='disc',status='unknown',form='formatted')
      write(16,fmt=503)((dis(i,l),l=1,2),i=1,200)
      endfile(unit=16)
      close(unit=16)
503    format(200(2i10/))
      return
      end
```

```
      program connect2
      common /block3/jpoly
      common /block4/kpoly
      common /block5/dis
      integer i,jco,lco,jpoly(934,5),kpoly(934,5),dis(200,2)
      character link*6,linkf*6,disc*6
      call readlink(link)
      call readisc(disc)
      i=0
      do 12 jco=1,934
      if(jpoly(jco,1).ne.0) then
      i=i+1
      kpoly(i,1)=jpoly(jco,1)
      do 13 j=1,200
      if(jpoly(jco,1).eq.dis(j,1))then
      if(dis(j,2).eq.jpoly(jco,2)) then
      jpoly(jco,2)=0
      else if(dis(j,2).eq.jpoly(jco,3)) then
      jpoly(jco,3)=0
      else if(dis(j,2).eq.jpoly(jco,4)) then
      jpoly(jco,4)=0
      else if(dis(j,2).eq.jpoly(jco,5)) then
      jpoly(jco,5)=0
      end if
      else if(jpoly(jco,1).eq.dis(j,2))then
      if(dis(j,1).eq.jpoly(jco,2)) then
      jpoly(jco,2)=0
      else if(dis(j,1).eq.jpoly(jco,3)) then
      jpoly(jco,3)=0
      else if(dis(j,1).eq.jpoly(jco,4)) then
      jpoly(jco,4)=0
      else if(dis(j,1).eq.jpoly(jco,5)) then
      jpoly(jco,5)=0
      end if
      end if
13    continue
      kpoly(i,2)=jpoly(jco,2)
      kpoly(i,3)=jpoly(jco,3)
      kpoly(i,4)=jpoly(jco,4)
      kpoly(i,5)=jpoly(jco,5)
      call writlink(linkf)
      end if
12    continue
      stop
      501  format(934(5i6/))
502   format('CONECT',5i6)
      end
      subroutine readlink(link)
      common /block3/jpoly
      integer jpoly(934,5),jco,lco
      character link*(*)
      open(14,file='link',status='unknown',form='formatted')
      rewind(unit=14)
      read(14,fmt=501,end=22)((jpoly(jco,lco),lco=1,5),jco=1,934)
```

```
22     close(unit=14)
501    format(934(5i6/))
       return
       end
       subroutine writlink(linkf)
       common /block4/kpoly
       integer kpoly(934,5),jco,lco
       open(15,file='linkf',status='unknown',form='formatted')
       do 23 jco=1,934
       write(15,fmt=502)kpoly(jco,1),kpoly(jco,2),kpoly(jco,3),
       +             kpoly(jco,4),kpoly(jco,5)
23    continue
       endfile(unit=15)
       close(unit=15)
502   format('CONECT',5i6)
       return
       end
       subroutine readisc(disc)
       common /block5/dis
       integer m
       parameter (m=200)
       integer dis(m,2)
       character disc*(*)
       open(16,file='disc',status='unknown',form='formatted')
       rewind(unit=16)
       read(16,fmt=503,end=33)((dis(i,l),l=1,2),i=1,200)
33      close(unit=16)
503     format(200(2i10/))
       return
       end
```

```
      program dumbel
      common /block1/ rx,ry,rz,ex,ey,ez
      integer n,natom
      parameter (n=100,natom=2)
      real rx(n),ry(n),rz(n),ex(n),ey(n),ez(n)
      real drmax,dens,sigma,dab(natom),dotmin,d
      real acm,accept,pi,ratio,rmin,rcut,deltv,deltvb
      real rxiold,ryiold,rziold,rxinew,ryinew,rzinew
      real exiold,eyiold,eziold,exinew,eyinew,ezinew
      real v,vnew,vold,vlast,vn,vs,temp
      real vlrc,vlrc6,vlrc12,control,beta
      integer step,i,nstep,iprint,isave,iratio
      logical ovrlap
      character cnfile*10
      parameter (pi=3.1415927)
      read input data********************
      open (12,file='inputdata',status='old',form='formatted')
c*****write input data*****
      read(12,'(i10 )')nstep
      read(12,'(i10 )')iprint
      read(12,'(i10 )')isave
      read(12,'(i10 )')iratio
      read(12,'(f10.4 )')temp
      read(12,'(f10.4 )')dens
      read(12,'(f10.4 )')rmin
      read(12,'(f10.4 )')drmax
      read(12,'(f10.4 )')tvbmax
      read(12,'(f10.4 )')rcut
      read(12,'(f10.4 )')d
      read(12,'(e12.4 )')control
      read(12,'(a )')cnfile
      write(*,'('' end of inputdata '')')
c*****convert input data to program units****
      beta=1.0/temp
      sigma =(dens/real(n))**(1.0/3.0)
      dab(1)=d*sigma/2
      dab(2)=-dab(1)
      drmax =drmax*sigma
      dotmin=0.2
      rmin=rmin*sigma
      rcut=rcut*sigma
      denslj=dens
      dens=dens/(sigma**3)
      if(rcut.gt.0.5)stop' cutoff too large '
      acm=0.0
      accept=0.0
      acv=0.0
c*****calculate long range correction
      sr3=(sigma/rcut)**3
      sr9=sr3**3
      vlrc12=8.0*pi*denslj*real(n)*sr9/9.0
```

```
      vlrc6 =8.0*pi*denslj*real(n)*sr3/3.0
      vlrc=vlrc12-vlrc6
c****write out information****
      write(*,'(''sigma/box  = '',f12.6)')sigma
      write(*,'(''dab(1)/box   = '',f12.8)')dab(1)
      write(*,'(''dab(2)/box   = '',f12.8)')dab(2)
      write(*,'(''drmax/box    = '',f12.6)')drmax
      write(*,'(''dotmin       = '',f12.6)')dotmin
      write(*,'(''rmin/box     = '',f12.6)')rmin
      write(*,'(''rcut/box     = '',f12.6)')rcut
      write(*,'(''tvbmax/box   = '',f12.6)')tvbmax
      write(*,'(''lrc for v    = '',f12.6)')vlrc
      write(*,'(''control      = '',e12.4)')control
      call readcn(cnfile)
c*****calculates initial energy and cheks for overlap**
      call check(rcut,rmin,sigma,dab,ovrlap,v)
      if(ovrlap)stop 'ovrlap in initial con '
      vs=(v+vlrc)/real(n)
      write(*,'(''initial v  = '',e20.6 )')vs
      write(*,'(/'' start of markov chain  ''/)')
      write(*,'('' acm      ratio       v/n  ''/)')
c*****loops over all cycles and all molecules  ********
      do 100 step=1,nstep
       do 99 i=1,n
          rxiold=rx(i)
          ryiold=ry(i)
          rziold=rz(i)
      exiold=ex(i)
      eyiold=ey(i)
      eziold=ez(i)
c*****calculates the energy of i in the old config
      call test(rcut,rmin,rxiold,ryiold,rziold,i,exiold,
      *          eyiold,eziold,sigma,dab,vold)
c*****move i and pickup the central image*****
      iseed=12349
      rxinew=rxiold+(2.0*ran(iseed)-1.0)*drmax
      ryinew=ryiold+(2.0*ran(iseed)-1.0)*drmax
      rzinew=rziold+(2.0*ran(iseed)-1.0)*drmax
      if (rxinew.gt.0.5) then
       rxinew=rxinew-1.0
      else if (rxinew.lt.-0.5) then
       rxinew=rxinew+1.0
      end if
      if (ryinew.gt.0.5) then
      ryinew=ryinew-1.0
      else if (ryinew.lt.-0.5) then
      ryinew=ryinew+1.0
      end if
      if (rzinew.gt.0.5) then
       rzinew=rzinew-1.0
      else if (rzinew.lt.-0.5) then
       rzinew=rzinew+1.0
      end if
c*****calculate the energy of i in the new config****
```

```
      call orien(exiold,eyiold,eziold,dotmin, exinew,
      *           eyinew,ezinew)
c*****check for acceptence***
      call test(rcut,rmin,rxinew,ryinew,rzinew,i,exinew,
      *          eyinew,ezinew,sigma,dab,vnew)
      deltv=vnew-vold
      deltvb=beta*deltv
      if(deltvb.lt.tvbmax) then
     if(deltv.le.0.0) then
     v=v+deltv
      rx(i)=rxinew
      ry(i)=ryinew
      rz(i)=rzinew
      ex(i)=exinew
      ey(i)=eyinew
      ez(i)=ezinew
      accept=accept+1.0
      else if (exp(-deltvb).gt.ran(iseed)) then
      v=v+deltv
      rx(i)=rxinew
      ry(i)=ryinew
      rz(i)=rzinew
      ex(i)=exinew
      ey(i)=eyinew
      ez(i)=ezinew
      accept=accept+1.0
      endif
      endif
99    continue
      acm=acm+1.0
      vn=(v+vlrc)/real(n)
      acv=acv+vn
c*******perform periodic operations
      if (mod(step,iratio).eq.0) then
      print *,'accept=',accept
      ratio=accept/real(n*iratio)
      if (ratio.gt.0.5) then
         drmax=drmax*1.05
     dotmin=dotmin*1.025
      else
         drmax=drmax*0.95
     dotmin=dotmin*0.975
      end if
      accept=0.0
      end if
      if (mod(step,iprint).eq.0) then
      write (*,'(i8,e14.4,e20.6)') int(acm),ratio,vn
      end if
      if (mod(step,isave).eq.0) then
       call writcn (cnfile)
      end if
      call readcn (cnfile)
100    continue
      write(*,'(//'' end of markov chain ''//)')
```

```
      call check (rcut,rmin,sigma,dab,ovrlap,vlast)
      if(ovrlap) stop 'overlap in final configuration'
      if(abs(vlast-v).gt.control) then
      write(*,'('' problem with energy ,'')')
      write(*,'(''vlast = '',e20.6)')vlast
      write(*,'('' v    = '',e20.6)')v
      endif
      call writcn (cnfile)
      avv=acv/acm
      write(*,'(/'' average ''/)')
      write(*,'(''<v/n> = '',e20.6)')avv
500    format(e10.4,t12,e10.4,t24,e10.4,t36,e10.4,t48,e10.4,
     *         t60,e10.4)
      stop
      end
      subroutine orien(exiold,eyiold,eziold,dotmin,
     *                    exinew,eyinew,ezinew)
      real exiold,eyiold,eziold,exinew,eyinew,ezinew,dotmin
      real dot,xi1,xi2,xi,xisq
      dot=0.0
100    if ((1.0-dot).ge.dotmin) then
      xisq=1.0
200    if(xisq.ge.1.0) then
      xi1=ran(iseed)*2.0-1.0
      xi2=ran(iseed)*2.0-1.0
      xisq=xi1*xi1+xi2*xi2
     goto 200
      endif
      xi=sqrt(1.0-xisq)
      exinew=2.0*xi1*xi
      eyinew=2.0*xi2*xi
      ezinew=1.0-2.0*xisq
      dot=exinew*exiold+eyinew*eyiold+ezinew*eziold
      goto 100
      endif
      return
      end

      subroutine test(rcut,rmin,rxi,ryi,rzi,i,exi,eyi,ezi,sigma,
     *                   dab,v)
      common /block1/rx,ry,rz,ex,ey,ez
c       calculates the total potential energy
c       integer n               the number of atoms
c       real rx,ry,rz           the positions of the atoms
c       real v                   the potential energy
c       real w                  the virial
c       logical overlap         true for substantial atom overlap
c         returns the total potential energy at the begining and
c         at the end of the run
c       ************************************************************
      integer n,natom
      parameter (n=100,natom=2)
      real rx(n),ry(n),rz(n)
      real ex(n),ey(n),ez(n)
```

```
      real rxi,ryi,rzi,exi,eyi,ezi,sr2,sr6
      real sigma,dab(natom),v,rcutsq,rminsq
      logical overlap
      real rxij,ryij,rzij,exj,eyj,ezj,vab
      real rxab,ryab,rzab,dabi,sigsq,rabsq
      integer i,j,ia,jb
c         ********************************************************
      overlap=.false.
      rcutsq=rcut*rcut
      rminsq=rmin*rmin
      sigsq=sigma*sigma
      v=0.0
c            ** loop over all the pairs in the liquid**
      do 100 j=1,n
      if(j.ne.i) then
      exj=ex(j)
      eyj=ey(j)
      ezj=ez(j)
      rxij=rxi-rx(j)
      ryij=ryi-ry(j)
      rzij=rzi-rz(j)
c        ** minimum image pair separations***
      if( rxij.gt.0.5) then
      rxij=rxij-1.0
      else if(rxij.lt.-0.5) then
      rxij=rxij+1.0
      end if
      if( ryij.gt.0.5) then
      ryij=ryij-1.0
      else if(ryij.lt.-0.5) then
      ryij=ryij+1.0
      end if
      if( rzij.gt.0.5) then
      rzij=rzij-1.0
      else if(rzij.lt.-0.5) then
      rzij=rzij+1.0
      end if
      do 99 ia=1,natom
      dabi=dab(ia)
      do 98 jb=1,natom
       rxab=rxij+exi*dabi+exj*dab(jb)
       ryab=ryij+eyi*dabi+eyj*dab(jb)
       rzab=rzij+ezi*dabi+ezj*dab(jb)
      rabsq=rxab*rxab+ryab*ryab+rzab*rzab
      if (rabsq.lt.rcutsq) then
      sr2=sigsq/rabsq
      sr6=sr2*sr2*sr2
      vab=sr6*(sr6-1.0)
      v=v+vab
      endif
98     continue
99     continue
      endif
100    continue
```

```
      v=4.0*v
      return
      end
      subroutine check(rcut,rmin,sigma,dab,ovrlap,v)
      common /block1/rx,ry,rz,ex,ey,ez
c       calculates the total potential energy
c       integer n               the number of molecules
c       real rx,ry,rz           the positions of the atoms
c       real v                   the potential energy
c       real w                  the virial
c       logical overlap         true for substantial atom overlap
c         returns the total potential energy at the begining and
c         at the end of the run
c       ************************************************************
      integer n,natom
      parameter (n=100,natom=2)
      real rx(n),ry(n),rz(n)
      real ex(n),ey(n),ez(n)
      real sigma,dab(natom),vab,sr2,sr6
      logical ovrlap
      real rxi,ryi,rzi,rxij,ryij,rzij,exi,eyi,ezi
      real exj,eyj,ezj,rxab,ryab,rzab,dabi,sigsq,rabsq
      integer i,j,ia,jb
c         ********************************************************

      overlap=.false.
      rcutsq=rcut*rcut
      rminsq=rmin*rmin
      sigsq=sigma*sigma
      v=0.0
      do 100 i=1,n-1
      rxi=rx(i)
      ryi=ry(i)
      rzi=rz(i)
      exi=ex(i)
      eyi=ey(i)
      ezi=ez(i)
      do 99 j=i+1,n
      rxij=rxi-rx(j)
      ryij=ryi-ry(j)
      rzij=rzi-rz(j)
c        ** minimum image pair separations***
      if( rxij.gt.0.5) then
      rxij=rxij-1.0
      else if(rxij.lt.-0.5) then
      rxij=rxij+1.0
      end if
      if( ryij.gt.0.5) then
      ryij=ryij-1.0
      else if(ryij.lt.-0.5) then
      ryij=ryij+1.0
      end if
      if( rzij.gt.0.5) then
      rzij=rzij-1.0
```

```
      else if(rzij.lt.-0.5) then
      rzij=rzij+1.0
      end if
      exj=ex(j)
      eyj=ey(j)
      ezj=ez(j)
      do 98 ia=1,natom
      dabi=dab(ia)
      do 97 jb=1,natom
        rxab=rxij+exi*dabi+exj*dab(jb)
        ryab=ryij+eyi*dabi+eyj*dab(jb)
        rzab=rzij+ezi*dabi+ezj*dab(jb)
      rxabsq=rxab*rxab
      ryabsq=ryab*ryab
      rzabsq=rzab*rzab
      rabsq=(rxabsq+ryabsq+rzabsq)
      if (rabsq.lt.rminsq) then
      ovrlap=.true.
      return
      else if(rabsq.lt.rcutsq) then
      sr2=sigsq/rabsq
      sr6=sr2*sr2*sr2
      vab=sr6*(sr6-1.0)
      v=v+vab
      end if
 97    continue
 98    continue
 99    continue
 100   continue
      v=4.0*v
      return
      end
      subroutine readcn (cnfile )
 C    ** SUBROUTINE TO READ IN THE CONFIGURATION FROM UNIT 10          **
      common /block1/rx,ry,rz,ex,ey,ez
      integer n
      parameter (n=100)
      real rx(n),ry(n),rz(n),ex(n),ey(n),ez(n)
      character cnfile*(*)
      integer cnunit
      parameter (cnunit=11)
      open(unit=cnunit,file='cnfile',status='old',form='formatted')
      rewind(unit=cnunit)
      do 30 i=1,n
      read (unit= cnunit,fmt=500,end=31) rx(i), ry(i),rz(i),
      *           ex(i),ey(i),ez(i)
      write(*,fmt=500)rx(i),ry(i),rz(i),ex(i),ey(i),ez(i)
 30    continue
 31    write(*,'(''end of file read'')')
      close (unit=cnunit)
 500    format(e10.4,t12,e10.4,t24,e10.4,t36,e10.4,t48,e10.4,
*          t60,e10.4)
      return
      end
```

```
      subroutine writcn (cnfile)
      common /block1/rx,ry,rz,ex,ey,ez
     i nteger n
      parameter (n=100)
      character cnfile*(*)
      real rx(n),ry(n),rz(n),ex(n),ey(n),ez(n)
      integer  cnunit
      parameter (cnunit=11)
      open (unit=cnunit,file='cnfile',form='formatted')
      do 40 i=1,n
      write(cnunit,fmt=500)rx(i),ry(i),rz(i),ex(i),ey(i),ez(i)
40     continue
      endfile (unit=cnunit)
      close (unit=cnunit)
500    format(e10.4,t12,e10.4,t24,e10.4,t36,e10.4,t48,e10.4,
     *         t60,e10.4)
      return
      end
```

LIST OF REFERENCES

REFERENCES

1. P. J. Flory, J. Am. Chem. Soc., **63**, 3083 (1941)

2. P. J. Flory, J. Am. Chem. Soc., **63**, 3091 (1941)

3. P. J. Flory, J. Am. Chem. Soc., **63**, 3096 (1941)

4. W. H. Stockmayer, J. Chem. Phy., **12**, 125 (1944)

5. M. Daoud and J. E. Martin, in The Fractal Approach to Heterogeneous Chemistry, Edited by D. Avnir (John Wiley, 1989)

6. C. W. Macosko and D. R. Miller, Macromolecules, **9**, 199 (1976)

7. D. R. Miller and C. W. Macosko, Macromolecules, **9**, 206 (1976)

8. R. J. J. Williams, Macromolecules, **21**, 2568 (1988)

9. R. J. J. Williams and C. I. Vallo, Macromolecules, **21**, 2571 (1988)

10. M. Gordon, Proc. Roy. Soc. London, **A268**, 240 (1962)

11. A. G. Mikos, C. G. Takoudis and N. A. Peppas, Macromolecules, **19**, 2174 (1986)

12. A. G. Mikos and N. A. Peppas, J. Controlled Release, **5**, 53 (1986)

13. H. Tobita and A. E. Hamielec, Macromolecules, **22**, 3098 (1989)

14. J. L. Jacobs and A. B. Scranton, Polymer News, **19**, 3, 81 (1994)

15. D. stauffer, A. Coniglio and M. Adam, Advances in polymer Science, **44**, 103 (1982)

16. P. Manneville and L. De Seze, in Numerical Mrthods in the study of Critical Phenomena, Edited by I. Della Dora, J. Demongeot and B. Lacolle (Springer, Berlin,1981)

17. H. J. Herrmann, D. Stauffer and D. P. Landau, J. Phys. A: Math. Gen., **16**, 1221 (1983)

18. R. Bansil, H. J. Herrmann and D. Stauffer, Macromolecules, **17**, 998 (1984)

19. H. M. J. Boots and N. A. Dotson, Polymer Communications, **29**, 346 (1988)

20. C. N. Bowman and N. A. Peppas, Chemical engineering Science, **47**, 1411 (1992)

21. M. P. Allen and D. J. Tildesley, Computer Simulation of liquids, Oxford Pub., (1987)

22. N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth and A. H. Teller, J. Chem. Phys., **21**, 1087 (1953)

23. V. M. Jansoone, Chem. Phys., **3**, 78 (1974)

24. Von Numann, US Nat. Bur. Stand. Appl. Math. Ser., **12**, 36 (1951)

25. D. Rigby and R. J. Roe, J. Chem. Phys., **87**, 7285 (1987)

26. R. Khare, M. E. Paulatis and S. R. Lustig, Macromolecules, **26**, 7203 (1993)

27. S. Gupta, G. B. Westermann-Clark, and I. Bitsanis, J. Chem. Phys., **98**, 634 (1993)

28. H. S. Kang, C. S. Lee, and F. H. Ree, J. Chem. Phys., **82**, 414 (1985)