



ELSEVIER

Computer Physics Communications 150 (2003) 65–84

Computer Physics
Communications

www.elsevier.com/locate/cpc

A parallel algorithm for molecular dynamics simulation of branched molecules

A. Jabbarzadeh *, J.D. Atkinson, R.I. Tanner

School of Aerospace, Mechanical and Mechatronic Engineering, The University of Sydney, NSW 2006, Australia

Received 14 January 2002

Abstract

To get an insight into the effects of molecular architecture in the behaviour of thin lubricant films we have devised an algorithm for simulation of branched molecules. We have used this algorithm successfully to simulate branched isomers of C_{30} . However the algorithm is flexible enough to be used for the simulation of more complex branched molecules. The resulting algorithm can be used in molecular dynamics simulation of branched molecules and could be helpful in designing new materials at the molecular level.

© 2002 Elsevier Science B.V. All rights reserved.

Keywords: Molecular dynamics; Branched molecules computer simulation; Molecular structure; Boundary lubrication and parallel processing

1. Introduction

Computer simulation of liquids has been an important tool in understanding the nature of flow and investigating the static and dynamic properties of the fluid at molecular scales. These simulations help researchers to understand how the molecules which constitute the liquid behave in different situations. The properties of liquids in some cases cannot be measured by experimental means. This can be due to the extreme conditions at which one wishes to measure the properties of the fluid. As an example, the lubricant in some parts of engines or in data storage devices may be sometimes exposed to extremely high shear rates as high as 10^9 s^{-1} [1]. These conditions can be simulated through non-equilibrium molecular dynamics (NEMD) computations. Understanding the effect of molecular structure on the properties of liquids in general and polymers and lubricants in particular is an active area of research [2–4]. A computational attack on this area through molecular level simulations requires developing efficient algorithms. A starting point would be understanding the effect of molecular branching on rheological properties of these liquids.

MD simulation requires the calculation of the interaction forces between the interaction sites, which are the primary elements of the liquid. These sites can be a single atom or a collection of atoms in a group such as CH, CH_2 or CH_3 . To reduce the computational effort a united atom model (UAM) is usually used and groups of

* Corresponding author.

E-mail address: ahmadj@mech.eng.usyd.edu.au (A. Jabbarzadeh).

atoms are considered as the interaction sites. Because of the short range forces used in most MD simulations the number of calculations can be significantly reduced by using special techniques such as building a neighbor list of interaction sites and using the link-cells method. The computational effort in MD simulations directly depends on the number of atoms in the simulation box. However, the level of complexity for MD algorithms also depends on the complexity of molecules. For simulations that involve only soft Lennard–Jones (LJ) particles calculation [5,6] of forces are straightforward. However for a realistic linear chain molecule [7,8] the level of complexity is a step higher as we have to calculate intramolecular interactions in addition to the LJ interactions between elements of different molecules.

Simulating branched molecules adds one more step to the complexity of the algorithm. This complexity arises mainly because of non-linear nature of the branched molecules and in the calculation of intramolecular interactions. Here in this work we will explain how we have handled these problems. Our work can be helpful for the researchers who are in the process of developing codes for simulation of similar molecules. We have employed a realistic model for the molecules and have shown how we have implemented this model to our existing algorithm. The assumption is that the branching is only on one level so there are branches off the main backbone of the molecules and the branches are not branched themselves. That is, the molecules are not hyperbranched.

The intramolecular potential parameters are well known for alkane molecules which have been widely used in MD simulations. A detailed model is used by Siepmann et al. [9,10]. This model gives results in agreement with experiments for alkanes.

We have built this algorithm upon our existing code [11] that is designed to simulate linear chain molecules on a multiprocessor environment using Parallel Virtual Machine (PVM) message passing software. This work explains the techniques used to simulate branched molecules using this parallel message-passing algorithm. Literature discussing such techniques and details is scarce and information provided here could be very helpful for researchers who wish to develop their own code for research in this field. Our intention was to use this code for fairly short molecules. However it has also proved to be effective for longer molecules. Although the information here is intended for simulation of molecules with one level of branching the framework introduced here can be extended to accommodate hyper-branched molecules as well.

2. Simulation details

Our molecular dynamics simulation algorithm is developed mainly for the study of thin liquid films confined between two solid atomic walls. Moving the walls in opposite directions simulates the lubrication process and generates Couette shear flow. A snapshot of a typical simulation box can be seen in Fig. 1.

2.1. The walls

Each wall is comprised of three layers of atoms of a BCC (body centred cubic) lattice. Each atom on the wall is attached by a stiff spring to its lattice position. The wall springs have a potential of the form

$$\phi_s = \frac{1}{2}k_w R^2, \quad (1)$$

where k_w is the spring stiffness and R is the distance of the wall atom from its lattice site. Periodic boundary conditions are applied in the x - and y -directions only. In addition a structureless wall inside the atomic wall, which has a very short range, is used to prevent the fluid molecules from penetrating inside the wall [12].

2.2. Model liquid

A united atom model is used to model the alkane molecules. In this model, groups of CH, CH₂ and CH₃ are treated as single interaction sites. That is, carbon–hydrogen interactions are ignored by adjusting the mass and

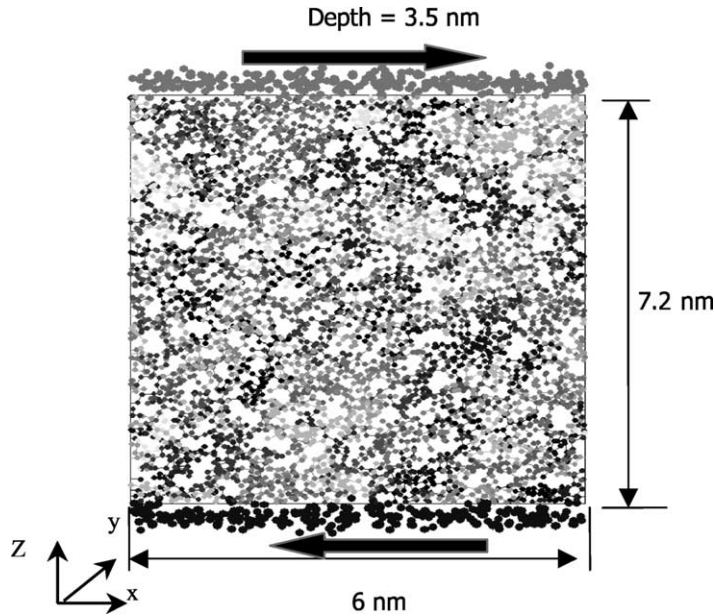


Fig. 1. Simulation box for Couette shear flow of $C_{12}H_{20}(C_3H_7)_6$ molecules. Molecules are shown with different shades to enhance the clarity of the snapshot.

other parameters to compensate it. We will refer to these groups as atoms for simplicity. The Lennard–Jones (LJ) potential given by Eq. (2) (below) governs the interactions of the atoms belonging to different molecules and also those of the atoms on the same molecule separated by more than three atoms.

$$\phi_{LJ}(r) = 4\varepsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r} \right)^{12} - \left(\frac{\sigma_{ij}}{r} \right)^6 \right] - \phi_{\text{shift}}, \quad (2)$$

$$\phi_{\text{shift}} = 4\varepsilon \left[\left(\frac{\sigma}{r_c} \right)^{12} - \left(\frac{\sigma}{r_c} \right)^6 \right].$$

The parameters we have used have proven to produce results in good agreement with experiments for liquid–vapor coexistence curves for linear and branched alkanes [9,13]. In this model, the Lennard–Jones energy parameter for methyl groups at the end of backbones is different from that for methyl groups at the end of branches. Also different LJ parameters are used for a tertiary carbon CH group at the branch sites from those of a CH_2 group. These parameters for LJ interaction potentials are given in Table 1. For the interaction of unlike groups Lorentz–Berthelot’s combining rules are used so $\varepsilon_{ij} = (\varepsilon_i \varepsilon_j)^{1/2}$ and $\sigma_{ij} = (\sigma_i + \sigma_j)/2$. For wall atoms the energy parameter is four times the energy parameter of CH_2 groups. This gives a value of 188 K for ε_w the energy parameter for wall– CH_2 interactions which is close to a typical surface energy of metals. For a gold surface ε_w is about 220 K [14], and for other metal surfaces values are typically in the same range.

Intramolecular architecture including bond stretching, angle bending and torsional potentials are included in the model. These potentials are given respectively by the following equations.

$$\phi(r) = \frac{1}{2}k(r_{ij} - r_0)^2, \quad (3)$$

$$\phi(\theta) = \frac{1}{2}k_\theta(\cos \theta - \cos \theta_0)^2, \quad (4)$$

Table 1
Parameters for the intramolecular and intermolecular interaction potentials

(A) LJ Potential	ϵ/k_B (K)	σ (nm)
CH ₃ , end group on the backbone	114	0.393
CH ₃ , side group	78	0.393
CH ₂ all methylene groups	47	0.393
CH groups	32	0.385
Wall atoms	752	0.393
(B) Bending angle potential $K_\theta = 520$ kJ/mol, $\theta_0 = 114^\circ$	(B) Bending angle potential at CH sites $K_\theta = 520$ kJ/mol, $\theta_0 = 112^\circ$	
(C) Stretching potential $K = 51600 \epsilon\sigma^{-2}$, $r_0 = 0.154$ nm		
(C) Torsional potential X–CH ₂ –CH ₂ –Y (kJ/mol) $C_0 = 9.2789$ $C_1 = 12.1557$ $C_2 = 13.1201$ $C_3 = -3.0597$ $C_4 = 26.2403$ $C_5 = -31.4950$	(C) Torsional potential X–CH–CH ₂ –Y (kJ/mol) $C_0 = 3.4070$ $C_1 = 7.5003$ $C_2 = 1.6281$ $C_3 = -15.3732$	

$$\phi(\alpha) = \sum_i^5 C_i (\cos \alpha)^i. \quad (5)$$

The parameters for the intramolecular and also intermolecular potentials are given in Table 1. The equilibrium bond angle at a branch site of CH is slightly smaller than that for other groups. Also for torsional potential for X–CH₂–CH₂–Y (X and Y can be any group of atoms) interactions, the original Ryckaert–Bellemans [15] model is used since it is shown in [10] that the result does not change if the potential replaced by values used in Siepmann et al.'s model in Refs. [9,13]. For interactions that involve a branch site of tertiary carbon X–CH–CH₂–Y different parameters are used as prescribed in [9].

Examples of branched molecules used in our studies by this algorithm are five isomers of C₃₀ alkane with different degree of branching. 3D snapshots, molecular structure and the chemical name of these molecules are shown in Table 2.

Equations of motion were integrated by a leap-frog Verlet method using a domain decomposition parallel algorithm on a cluster of DEC Alpha 500 workstations by using PVM (Parallel Virtual Machine) message passing software [16]. The parallel algorithm is described in the following sections.

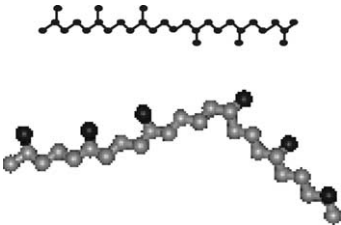
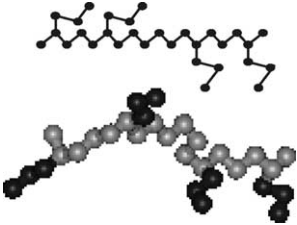
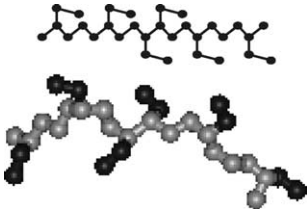
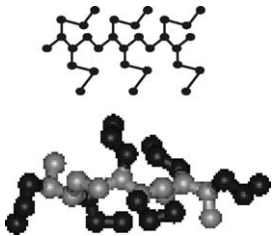
3. The parallel algorithm

3.1. Choosing a proper parallel computing strategy

In designing a parallel code for MD there are basically three options. One method is the replicated data (RD) [17] algorithm at which every processor has the identical data that defines the system. Each processor is then allocated a part of the force computation workload, and after completion the equations of motion are integrated independently on each processor. Although this method has a good work balance it is very wasteful of memory and is suitable for only small systems and shared memory computers [18]. Furthermore, for the RD model in some instances an increase in the number of processors can result in a decrease in the speedup [17]. Another method is the systolic loop (SLS-G) algorithm [19]. In this method each processor is assigned a group regardless of their positions. Then in each systolic cycle packets of data containing the atomic positions of the relevant atoms and force accumulators sent from each processor travel through all the other processors and on the return this data packet contains the completed force accumulator array. These data then are used for the integration of the equations of motion for the group of atoms in that processor. Despite its good load balancing this method is suitable only for systems where

Table 2

Molecular structure, chemical name and degree of branching for the alkane molecules simulated here. Branches are shown with a darker shade to improve the clarity

Molecular structure and a 3D snapshot	Chemical name	Generic name used here
	2,6,10,15,19,23,hexamethyltetracosane	Squalane
	2,6,13,17,tetrapropyloctadecane	$C_{18}(C_3)_4$
	2,6,9,10,13,17,hexaethyloctadecane	$C_{18}(C_2)_6$
	2,3,6,7,10,11,hexapropyldodecane	$C_{12}(C_3)_6$

long-range interactions are important. When the interactions between the particles of the system are short range compared to the system size the link-cells [17,19] method is the natural approach. The parallel link-cells method is both fast, and economical as far as the memory is concerned. In the present work we have adopted this method for our algorithm. However some modifications are necessary in order to apply it for chain and branched molecules in presence of structured walls. This will be discussed in the coming sections.

3.2. Communication strategy

The communication between the processors is handled through a message passing software called PVM. Based on the organization of the computing tasks the programming approach in PVM can use various models. One of the

most commonly used models is the crowd model. However other models such as tree and hybrid models are usually used depending on the problem [16]. In the crowd model a number of closely related processors execute the same program and do the computations on different parts of the workload. This method itself is usually implemented in two forms. The first is the master–slave or client–server model in which there is a separate program running on the master processor that controls the initialization, spawning, and collection and displaying the data. The other processors are usually called slaves or servers. They run a separate code and do the actual computation. The other model is the node-only model where multiple copies of a single program are executed in several processors. In this model the first copy of the program is initiated manually on the console. By checking the parent address of the task it is possible for the program to realize that it is the first copy. Then the parent processor where the first copy of program is started spawn the program on the other processors. The parent processor is usually assigned to take over the non-computational responsibilities and also to contribute to the computational workload. Our algorithm uses node-only model. PVM has the option to take care of the communications through TCP/IP ports and also through PVM daemon. These options can be easily utilized by calling relevant PVM subroutines.

3.3. Parallel link-cells method

In this method the simulation domain is divided into several subdomains and the atoms in each subdomain are assigned to a processor. Each processor is responsible only for the atoms in its subdomain. As soon as an atom leaves a subdomain all the information about that atom should be sent to the processor responsible for the subdomain in which the atom arrives. If the system is large enough most of the interactions happen among the atoms in the same subdomain. However we should take into the account the interactions that happen between the atoms in a given processor and the atoms in the neighboring subdomains assigned to other processors. Copying the positions of the possible interaction sites from the neighboring processors satisfies these requirements. For short range interactions candidates are in the boundary areas that surround the subdomain. These provisions ensure that LJ interactions are correctly and efficiently calculated. For bigger molecules such as linear or branched molecules we have to use a method that correctly deals with the contribution of the neighboring processor to the bonded interactions. We will describe this method in the coming sections.

3.4. Atomic labels

In order to distinguish the atoms, including both wall atoms and fluid atoms, each atom will have two identification numbers, which can be simply assigned to them. These ID numbers are assigned to the atoms in the initialization process when the system is being built up. These numbers should be unique for each atom regardless of their host processor. These ID numbers are stored in two arrays. The length of each array is the total number of wall and fluid atoms in the system. The first array contains the wall ID numbers and second array contains fluid ID numbers. The ID numbers in each array are simply their sequential numbers in the process of building. The fluid atoms are built first so the fluid ID numbers will be assigned to them only in the fluid ID array. Then the wall atoms are built and at this stage the numbers are assigned only to the array that contains wall ID numbers. At this point, in the fluid ID array, the first part contains the atomic IDs for the fluid atoms, and the second part elements are all zero. For the wall ID array the first part elements are all zero and second part elements are wall ID numbers. Then if we check the fluid ID number for a wall atom it will be zero and if we check the wall ID number for a fluid atom it will also be zero. This makes it very simple to distinguish between a wall and a fluid atom. The non-zero numbers in those arrays will be ID number of that atom (*atomID*). This number is unique for each atom and will be used as a reference to check for other information as needed. As an atom moves from one processor to the other it is essential we pass its *atomID* with other relevant information.

We should also know the parent molecule for each atom or interaction site. Each molecule has a unique ID number and the segments of each molecule are tagged with that ID number which is called *molid*. The way that an atom is tagged with its parent molecule ID is through a unique number. Here we can find another use for the fluid

atomic IDs. Since *atomID* is produced sequentially it is easy to find *molID* from *atomID*. The way this process is handled is slightly different for branched molecules, which will be described later. Having these ID numbers for the molecules and atoms make it much easier to use them in the process of simulation and when they move around from one processor to another.

3.5. Domain decomposition

A parallel link-list method in conjunction with the neighbor list method is employed to take full advantage of the short-range interactions. For a large system of particles huge computational times may be spent on building this neighbor list. This can be avoided if the link-lists method is employed in conjunction to the neighbor list method. The link-cells method is used to update the neighbor list approximately every 10–20 time steps. The updating time is determined by monitoring the maximum velocity in every time step. This ensures that no atom has left its cell before the list is updated. In the parallel version of the link-list method the simulation region is divided into subdomains and each subdomain is divided into a lattice of cells. Neither the subdomain nor the cells need to be cubic. The sides of these cells exceed the cut off radius r_c in length. So an atom in one of the cells can only interact with the atoms in the same cell or with the atoms in immediately adjacent cells. To ensure that the neighbor list built by the link-cells method remains valid for 10–20 successive time steps the side of the cells are chosen to be slightly larger than the cut off distance r_c . This small distance is called the neighbor list shell and for liquids a value of 0.3–0.4 in reduced units based on atomic diameter is typically used [18]. Fig. 2 shows the parallel link-cells method applied in two dimensions. The extension to a 3D simulation is straightforward and similar. Thick lines show the boundary for the domain of one processor. Dashed lines divide the domain to a number of link-cells. The data including positions, velocity and atomic ID numbers for the atoms in the cells marked with “C” are copied from the neighboring processors into the processor in the middle.

To avoid double counting of the interactions pairs one can see that in three dimensions it is necessary to search atoms only in 14 out of 27 neighboring cells. If N is the number of atoms in the system the link-cells method reduces the computational effort from $O(N^2)$ to $O(N)$. When determining the number of cells two extra cells should be considered in each direction (one at each side) for each processor. We will assign the copied atoms from the neighboring processors to these cells. In some cases a processor can be adjacent to a wall. The wall atoms in

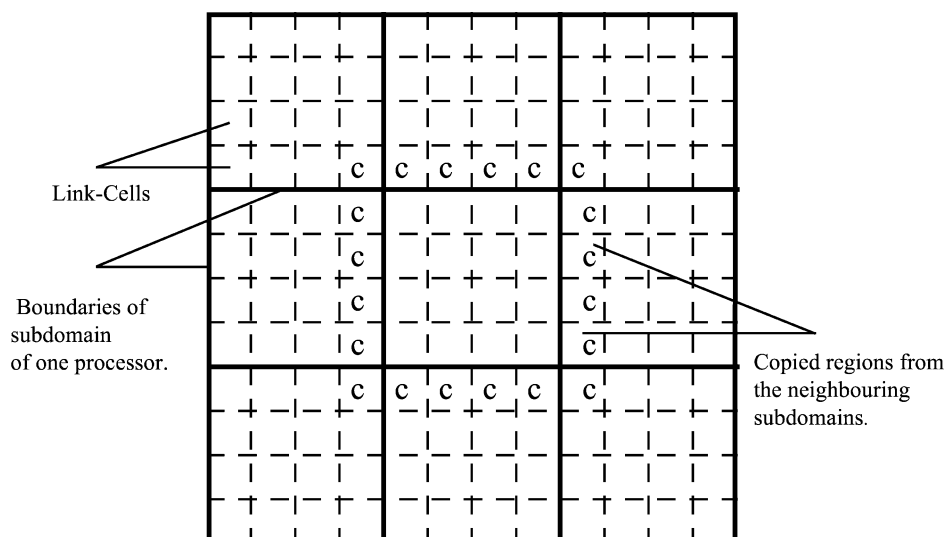


Fig. 2. Domain decomposition in two dimensions.

this processor may move slightly beyond the simulated region. In these cases the extra cells in the Z -direction are used to accommodate these atoms.

3.6. Periodic boundaries

The periodic boundary conditions are applied only in X - and Y -directions. In the Z -direction the walls are present and the boundaries are non periodic. Thus it is important to make sure that all the processors understand this periodicity. The first step in this process is to make sure that each processor knows its neighboring processors. This can be done by determining the neighboring processors in each direction (two in each direction) and store them in appropriate arrays. However for the processors that are located next to a wall there is no neighboring processor in the Z -direction at the side where it is adjacent to the wall. In these situations the array that contains the address for the neighboring processors should be given some number that cannot be mistaken with anything else—a negative integer for instance. So each time a processor checks for its neighbor in a certain direction and comes up with a negative number it will realize that there is no neighbor in that particular direction.

The application of the periodic boundary conditions will be done during the copying and moving operations between the processors. Fig. 3 shows an example of domain decomposition for eight processors. The number on the face of each cube shows the processor number. Table 3 shows the neighboring processors in all directions for this example.

The periodicity in the X - and Y -directions is implicitly included in setting the neighboring processors. So for instance if an atom leaves processor 2 from the positive side of the Y -axis it should be moved to processor 1. The co-ordinates of the atoms should be adjusted during the moving or copying operations according to the periodicity.

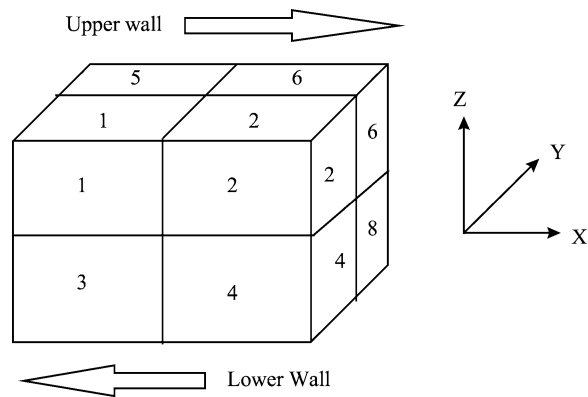


Fig. 3. This is an example of domain decomposition with 8 processors.

Table 3
Neighboring processors for the example shown in Fig. 3

Processor number	1	2	3	4	5	6	7	8
X -direction	2	1	4	3	6	5	8	7
$-X$ -direction	2	1	4	3	6	5	8	7
Y -direction	5	6	7	8	1	2	3	4
$-Y$ -direction	5	6	7	8	1	2	3	4
Z -direction	none	none	1	2	none	none	5	6
$-Z$ -direction	3	4	none	none	7	8	none	none

3.7. Bonded data lists

For the molecules, the bond interactions must be determined as well as the non-bond interactions. Information regarding the atomic positions is needed for the calculations of bond forces. Sometimes however a molecule will straddle two or more processors. So the information might not be completely available to the processor partly containing a molecule, because some of the atoms of the same molecule reside in different processors. To have the necessary data the first step is to create a bond data list (BDL). This list contains the *atomID* for the atoms of the molecules that completely or partly reside in the processor. To build a BDL we perform the following steps:

- (1) Search through the atoms that reside in each processor and make a list of the molecules that are completely or partly in that processor. Using the molecular IDs (*molID*) for these molecules, this list contains the information that points to the atoms that belong to these molecules.
- (2) Make a list of the incomplete molecules based on their *molID*.
- (3) Broadcast the list *molIDs* made in step 2 to all the other processors.
- (4) Receiving processors will search for the atoms that belong to the molecules in the list and pack their coordinates and atomic IDs and send them back to the inquiring processor.
- (5) The received data from the other processors are collected and used to complete the BDL list for all the atoms in the molecule.

The completed BDL in step 5 will contain *atomID* for the segments of molecules that reside entirely or partly in the processor. BDL is the main data list that will be used to calculate the intramolecular interactions in an organized and indexed fashion.

For linear chains of molecules BDL can be a simple two-dimensional array. The first dimension refers to the molecular ID and second dimension to the sequential position of the atom in the molecule along the backbone. The data stored in the BDL simply refer to the fluid atomic IDs (*atomID*) of the atoms in the molecule. For branched molecules this will be described later.

It should be mentioned that in the computation of the bonded interactions only those interactions should be computed that involve at least one of the atoms that actually reside in the processor. Although this might result in having some replicated data from other processors that might not be used in the computations it guarantees that all the information needed for these computations is in place. For large enough systems only a small fraction of the data will be duplicated in this form. The process for calculating the local properties such as velocity, density, stresses and temperature should be done globally over all the processors. The parent processor then calculates streaming velocity and broadcasts it to all the other processors if needed.

4. Implementing branched molecules

Simulating branched molecules add another complexity level to the algorithm that should be dealt with carefully. Having a more elaborate model where branched sites are treated differently from the ones on the backbone make it necessary to distinguish these sites from the others. Also in calculating intramolecular potentials especially for the torsional potential it is necessary to have techniques in place to distinguish between cases that involve branched sites namely where the tertiary carbon atom is (CH group). Also because of branching it is not possible to simply sweep along the backbone and calculate the intramolecular interactions. A more elaborate scheme should be employed to conduct correct and efficient calculations.

4.1. Introducing new parameters

In order to distinguish interaction sites on the branches and their location and track them for relevant calculations it is necessary to have some parameters in place. First of all we need to recognize the branch sites where tertiary

carbons are. So we define a parameter such as *BranchSite* that is an array of integers. Its length is equal to the total number of interaction sites of liquid molecules in the simulation. It will have value of zero if it is not a branched site (CH group), otherwise it will have the position number along the backbone. For example, for a molecule which has a length of 20 segments if a segment is CH group and located at position number 12 then *BranchSite* stores number 12. For any other groups (CH₂ or CH₃) it will have a zero value. *BranchSite* is referenced by *atomID* that is unique to each interaction site.

We also need to define a parameter that returns the branch site *atomID* for the atoms located on the branch. This will be necessary when we calculate the intramolecular interactions. This can be an array of integers of length equal to the total number of fluid atoms, called *branchSiteID*. Also it is necessary to know the position of each atom in the branch. A group containing tertiary carbon has the rank = 0 and the rest are numbered sequentially along that branch. The sequential positions of atoms along the same branch are stored in an array called *branchRank*. This array of integers has the length equal to the total number of fluid atoms in the system. Similar to other parameters mentioned here it is referenced by *atomID*.

The atoms on the branch including the CH site are recognized by using an array called *inBranch*. For non-branch sites on the backbone *inBranch* is zero. For atoms on the branch and for CH site it has the position number of branch along the backbone. We can use this array to check if an atom is on a branch or if the interaction sites are on the same branch.

4.2. Labeling of the atoms and Bonded Data List (BDL) for branched molecules

Labeling the *atomID* for the atoms of the fluid molecules for linear chain is sequential and is done one by one for the molecules along the backbone. For branched molecules however the procedure of labeling is done in two stages:

- (1) Label the atoms in the backbone first sweeping from one end to the other.
- (2) Sweep again along the backbone and if there is a branch sweep along it and label the atoms in the branch.
- (3) Continue from the next position left on the backbone and repeat step 2 until finished.

The procedure is shown for a typical branched molecule in Fig. 4.

Our branched molecules have only one level of branching, i.e. those branches do not divide in branches themselves and so on. So unlike linear chain molecules where the BDL is a two-dimensional array, for branched molecules a three-dimensional array should be allocated.

The dimensions of this array BDL should be (number of molecules, number of atoms on the backbone, number of atoms on the branch). BDL stores *atomID* for the segments of the molecule according to the molecule number, their position on the backbone and their position on the branch. So $BDL(n, i, j)$ stores *atomID* of the *j*th atom in the branch which is coming off the *i*th atom along the backbone of the *n*th molecule. The idea is to use these data

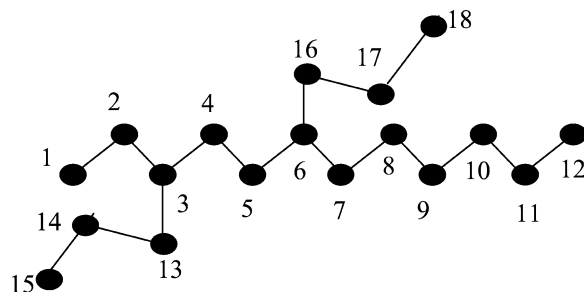


Fig. 4. Sequence used for labeling atomic IDs for a typical branched molecule.

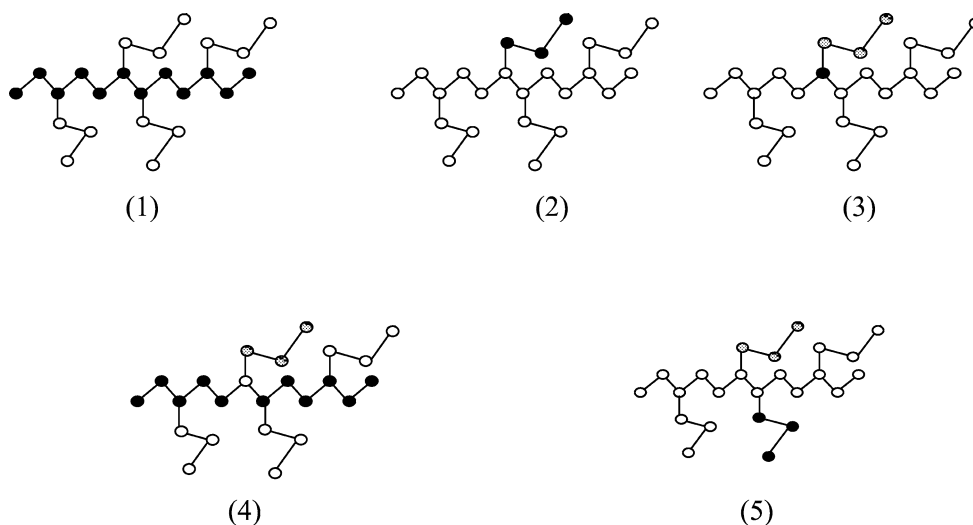


Fig. 5. Examples of the five distinct cases for calculation of LJ interactions for a typical branched molecule. For molecules where possible pairs of atoms are shown only in black any black–black interaction falls in that category. For molecules where pairs are shown with black and patterned backgrounds any black–patterned interaction fall under that category. Five cases are described above. Depending on the category the pairs should be checked to see if are separated enough to calculate LJ interaction.

to keep track of the atomic positions of all the segments of a molecule which is located on multiple processors. To build BDL for branched molecules we follow the same procedure as we did for linear molecules as described in Section 3.7. For branched molecules which have higher levels of branching a four- or five-dimensional array may be used following the same procedure and having the higher dimensions to record data for atoms at branches at higher levels. A high level of branching though could significantly add to the memory requirements for the BDL depending on the number of molecules and its branching pattern. We found that this algorithm works well for molecules with one level of branching for moderately short-branched molecules. We tested also branched molecules with a backbone length of 250 segments and did not find memory related problems. We recognize that using BDL in this form can result in some unnecessary memory usage but it greatly facilitates handling of intramolecular interactions. For our purposes, dealing with molecules with low levels of branching, this technique proved to be efficient.

4.3. Calculating intermolecular interactions

Lennard–Jones interactions are calculated for pairs of atoms that belong to different molecules, or pair of atoms that are on the same molecule but are separated by more than three atoms. The idea is to exclude pairs of atoms that interact via any intramolecular interaction given by Eqs. (3), (4) or (5) from LJ interactions. In a linear molecule this can be simply achieved by checking their sequential position along the backbone. In branched molecules since the atoms are not indexed sequentially new parameters that were introduced in Section 4.1 prove to be useful. For pair interactions of the atoms on the same molecule possible cases then are:

- (1) Both atoms are on the backbone where any of the following is true:
 - (a) Neither atom in the pair is on a branch (can be checked using *inBranch*).
 - (b) One of the atoms is on the backbone and one on a branch but it is a branch site (CH group).
- (2) Both atoms are on the same branch but neither is a branch site (CH group).

In these two cases then a check on the *atomId* would show if three or more atoms separate those atoms.

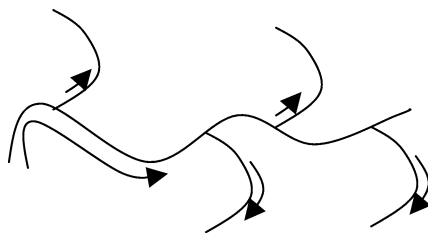


Fig. 6. The sweep pattern for calculating bond stretching potential.

- (3) Both atoms are on the same branch where one of them is a branch site (CH group). In this case a check on the position of the atoms on the branch by using *branchRank* would show if three or more atoms separate those atoms.
- (4) One of the atoms is on a branch and the other on the backbone where any of the following is true:
 - (a) One atom is on a branch another on the backbone and it is a non-branch site.
 - (b) Both are on a branch and one is a branch site (CH group).

In this case the head of the branch (CH group) should be used to check on the separation between the atoms. Parameter *branchSiteID*, which returns the *atomID* of the head of the branch (CH group), becomes handy in this case. So that the *atomID* of the atom on the backbone and the *atomID* of the CH group for the branch (found by *branchSiteID*) where the second atom is located should be separated by at least

$$3 - \text{branchRank}(\text{of the atom on the branch})$$

- (5) Two atoms are located on different branches and are non-CH group. Then the *atomID* of the CH group of the two branches should be separated by at least:

$$3 - \text{branchRank}(\text{first atom}) - \text{branchRank}(\text{second atom})$$

The mentioned steps can be programmed easily. A typical Fortran listing for this part is provided in Appendix A as a guide for interested programmers.

4.4. Calculating Intramolecular Interactions

In calculating intramolecular interaction for linear molecules it is fairly easy to sweep along the same molecule and calculate the intramolecular potentials such as stretching, bond angle and torsional potential and the forces. In that case since the indexing of the atomic ID numbers are sequential along the chain the task is more straightforward. However for branched molecules a more elaborate scheme should be devised.

As we mentioned before the BDL that contains the atomic ID of all the atoms that belong to the same molecule need to be built prior to calculation of intramolecular interactions.

4.4.1. Stretching potential

This potential is given by Eq. (3). We need to calculate the bond length between the atoms. For a linear molecule a sweep along the backbone using BDL list will do the job. Since the same potential parameters are used for all bonds there is not much difference in the case of branched molecules. In that case this is done in two stages.

- (1) Sweep along the backbone calculate the potential for $\text{BDL}(n, i, 1)$ and $\text{BDL}(n, i + 1, 1)$ and increment i one at a time until finished.
- (2) Go through the backbone and if encounter a branch sweep through that branch. Calculate the potential for $\text{BDL}(n, i, m)$ and $\text{BDL}(n, i, m + 1)$ and increment m one at a time until that branch is finished. Repeat the

same for all the other branches. So sweep along the backbone check if there is a branch and sweep through it. This can be coded elegantly using a few nested do loops. The listing for this is given in Appendix A.

4.4.2. Calculating bond angle potential

This potential is calculated using Eq. (4). This is a three body potential and depending on the atoms involved different parameters should be used. As we saw in Table 1 the bond angle is slightly smaller where the tertiary carbon atom is located.

A branched molecule is more complex than a linear molecule. For a linear molecule again a sweep through the BDL will be enough. In that case each time three consecutive atoms are considered for calculation and we go forward one atom at a time. For branch molecules however there is more complexity. This is done in two stages.

- (1) In first stage we treat it as a linear chain and sweep through the backbone of the molecule and calculate the bond angle potential and forces. We start from the beginning of the chain and consider three consecutive atoms using the BDL. We shift one atom at a time and finish through the backbone. So we do the calculation for the atoms $BDL(n, i, 1)$, $BDL(n, i + 1, 1)$ and $BDL(n, i + 2, 1)$ and increment i one at a time.
- (2) In second stage we again sweep through the backbone but only consider three atoms that include a branch site where CH group is. This sweep should be both forward and backward when we encounter a branch along the backbone. The provisions for this move are shown in Fig. 7. As we move along the backbone for a given atom along the backbone on a branch site two angle potentials interactions should be calculated as shown. The focus of attention is atom $BDL(n, i, 1)$. So one set will include $BDL(n, i, 2)$, $BDL(n, i, 1)$ and $BDL(n, i + 1, 1)$ and the other set includes $BDL(n, i, 2)$, $BDL(n, i, 1)$ and $BDL(n, i - 1, 1)$. As we sweep forward we change our focus to other CH sites and repeat the same procedure.
- (3) In third stage we sweep through the branches and consider only atoms that are completely on that branch.

This procedure can be coded easily. The listing of the code in Fortran language is given in Appendix A.

4.4.3. Calculating torsional potential

Torsional potential is a 4 body potential. It is calculated using Eq. (5). Again for linear chains it is done using BDL and sweeping along the backbone of molecule. We shift one atom as we sweep until at least four atoms can be included in the potential calculation.

For a branched molecule this is done in many stages.

- (1) In first stage we go through the backbone of the molecules. We calculate the torsional potential for the first four atoms. We shift one atom at a time and repeat the procedure until there are not enough atoms left.
- (2) In the second stage we sweep along the backbone and when we come along a branch if the branch is long enough (4 atoms or longer) we sweep through it and shift one atom at a time until there is less than four atoms to the end.

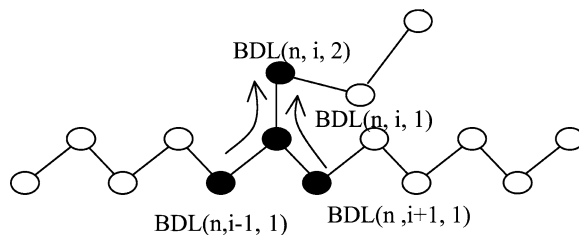


Fig. 7. Bond angle potential when involves the CH group is calculated as shown using BDL. Here n is the number of molecule, i is the number of molecule along the backbone. Atom $BDL(n, i, 1)$, a CH group, is the focus of attention and two angle interactions involving this atom are calculated.

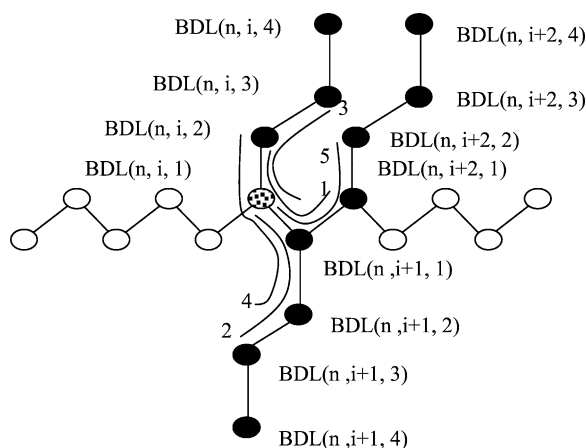


Fig. 8. The provision for calculation of torsional potential involving a CH group for a branched molecule. As we sweep forward along the backbone an atom such as $BDL(n, i, 1)$ in this case becomes the focus of our attention. There are maximum 5 different possible interactions involving this atom and other atoms on the branches next to it. The actual cases depend on the molecular structure. The interactions and the atoms involved are shown and numbered accordingly in this diagram.

(3) In the third stage we sweep along the backbone again. For any atoms on the backbone there are five provisions that provide for possible torsional interactions involving this atom and other atom or atoms on the nearby branches. These provisions are set in Fig. 8. These interactions involve a mixture of atoms on the backbone, on the branches and the CH site. As we go forward along the backbone of the molecule for any atom on the backbone depending on the molecule structure there are maximum 5 torsional potential interactions involving this site and the atoms on the branches. In this figure the atom $BDL(n, i, j)$ which is under examination is shown with dotted patterned background. Other sites involved in those interactions are shown in black and indexed through BDL list and are given below.

$$1 \begin{cases} (n, i, 2) \\ (n, i, 1) \\ (n, i+1, 1) \\ (n, i+2, 1) \end{cases} \quad 2 \begin{cases} (n, i, 1) \\ (n, i+1, 1) \\ (n, i+1, 2) \\ (n, i+1, 3) \end{cases} \quad 3 \begin{cases} (n, i, 3) \\ (n, i, 2) \\ (n, i, 1) \\ (n, i+1, 1) \end{cases} \quad 4 \begin{cases} (n, i, 2) \\ (n, i, 1) \\ (n, i+1, 1) \\ (n, i+1, 2) \end{cases} \quad 5 \begin{cases} (n, i, 1) \\ (n, i+1, 1) \\ (n, i+2, 1) \\ (n, i+2, 2) \end{cases}$$

For example an interaction involving

$$\begin{cases} (n, i-1, 1) \\ (n, i, 1) \\ (n, i, 2) \\ (n, i, 3) \end{cases}$$

will be considered when $(n, i-1, 1)$ is the focus of attention. This falls in category number (2) of the above provisions. Any other possible interaction falls in one of the five mentioned categories.

You can refer to Fig. 8 for position of the atoms along the molecule for understanding the indexing methodology used here. This method of indexing provides a convenient way to process the atoms in the BDL and calculate the torsional potential interaction easily. A combination of conditional statements and do loops examine for possible existing scenarios and conduct the required calculations. This procedure can be easily programmed as we did for the other potentials in Appendix A.

5. Performance analysis

5.1. Benchmarks

We tested the performance of our MD algorithm on a PVM farm consisting of 16 DEC Alpha 500 workstations that were connected by a 100 Mbits/sec Ethernet. We examined three systems with different sizes. For all the systems 2,3,6,7,10,11,hexapropyldodecane ($C_{12}(C_3)_6$) was used as the fluid. This is a short branched molecule with C_{12} backbone and six branches of C_3 oriented symmetrically on the backbone (see Table 2). The interaction cut off distance $r_c = 0.82$ nm was chosen. Three systems were simulated for performance tests as follow:

Small: A total number of 200 $C_{12}(C_3)_6$ molecules with 1008 atoms on the walls making a system of 7008 atoms.

Medium: 480 molecules of $C_{12}(C_3)_6$ with 1932 atoms on the walls. The total number of atoms was 16,332.

Large: 1600 $C_{12}(C_3)_6$ molecules with 3864 atoms on the walls. The total number of atoms was 51,864.

For all the simulated systems a shear rate of $1.78 \times 10^{10} \text{ s}^{-1}$ was applied. The isothermal shear flow was simulated at a temperature of 480 K.

5.2. Speedup and efficiency

Speedup and the efficiency indicate the performance of any parallel algorithm. Speedup shows how much faster the program runs on a multiprocessor system compared to the time it takes for it to run on a single machine. Speedup is usually measured by comparing the CPU time taken for the parallel code to run on a number of processors to the CPU time taken for the parallel algorithm to run on a single processor. The speed up is defined by (6)

$$S_p = \frac{T_s}{T_p}, \quad (6)$$

where S_p is the speedup and T_s and T_p respectively are the CPU time on a single processor and CPU time on a multiple processor system to run the parallel code. For multiprocessor system the average CPU time in usually used for T_p .

Another measure of the performance of a parallel algorithm is the efficiency. Efficiency measures scalability of the parallel code. Efficiency is given by

$$Eff = \frac{S_p}{N_p}, \quad (7)$$

where S_p is the speedup and N_p is the number of the processors. A 100% efficiency is obtained if the increase in the speedup has the same proportion as the increase in the number of processors. This means if we double the number of processors we get a speedup of 2. In practice however it is not feasible to have 100% efficiency. The CPU time spent on each processor consists of two parts. One part is the actual computation time and another part is for the communication that is mainly spent by the PVM daemon for executing the message passing routines. In any parallel algorithm the goal is to reduce the communication time as much as possible. To use Eqs. (6) and (7) one should run the parallel code on a homogeneous collection of processors so that the CPU time on the machines can be compared. Here we did use similar DEC ALPHA machines to satisfy this condition.

5.3. Performance of the benchmark systems

Using the benchmarks described in Section 5.1 we performed a number of test runs. For all these runs the duration of the simulation was 10,000 time steps (47 ps) excluding the start-up and initialization stages. All the runs started from the same configuration. Fig. 9 shows the speedup as a function of the number of processors. The

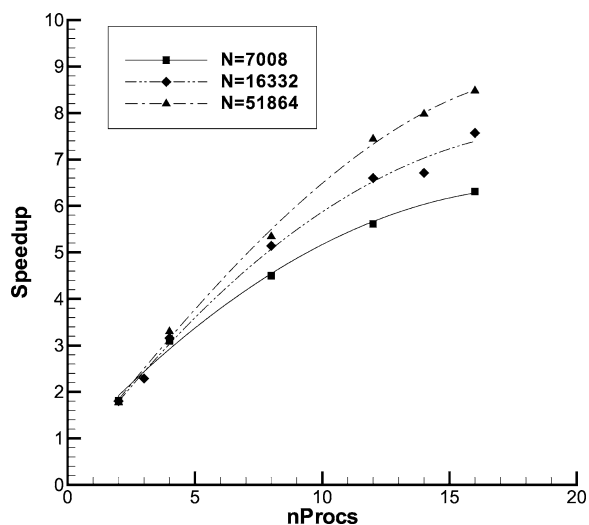


Fig. 9.

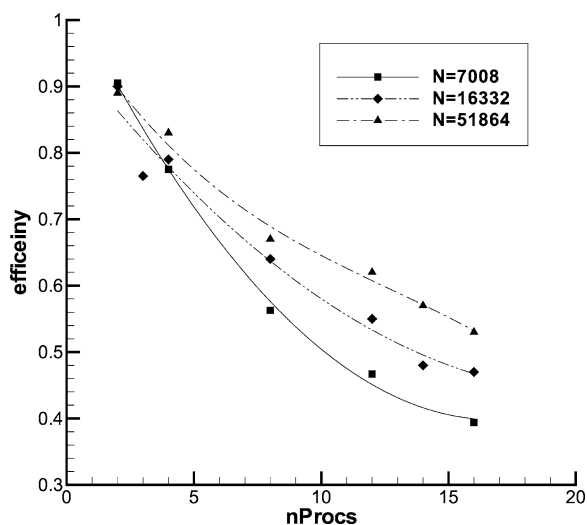


Fig. 10.

Fig. 9. Speedup against the number of processors for three benchmark systems. N is the number of atoms in the system.

Fig. 10. The efficiency of the parallel algorithm on PVM against the number of processors. N is the number of atoms in the system.

results are presented for three benchmark systems. Although the speedup is not linear as would be expected in an ideal situation it is an increasing function of the number of processors. Fig. 10 shows the efficiency of the parallel MD on the PVM. As it can be seen the efficiency has higher values with a smaller number of processors in the virtual machine. The efficiency decreases with increasing the number of processors.

For both the efficiency and speedup it can be seen that the results are very close for *small*, *medium* and *large* systems at lower number of processors. Though, with more processors increasing the system size increases both the efficiency and the speedup. This means that the algorithm is more efficient for larger systems.

The overall results for the speedup and efficiency are in a reasonable range and a maximum speedup of 8.5 can be reached by using 16 processors. The reason for the decrease in the efficiency is mainly because of an increase in the communication time between the processors. Fig. 11 shows the ratio of the communication time to the total CPU time against the number of processors. We can see this ratio increases with the number of processors in the virtual machine for all the benchmark systems. It can also be seen that this ratio is smaller for larger systems especially at systems with larger number of processors. Communication takes place while copying the atomic coordinates from the neighboring processors and also when atoms move from one subdomain to another. Although a copying operation takes place every time step, movement operation takes place every 10–20 time steps when the neighbor-list is rebuilt. There are other occasions when the communication takes place between the processors every time steps such as, calculating the thermostat, building BDL for bond interactions and calculating average properties. Among these operations the copying operation is probably the most time consuming operation followed by building the BDL. During the copying operation we need to send the required information for the integration of equations of motion. We used the Verlet algorithm that requires only two arrays (velocities and positions) of the same size in compare to 10 arrays for the Gear predictor-corrector method (velocities and positions and their first to fourth derivatives). This explains the lower efficiency for larger systems of hexadecane simulations in our earlier work [11], which used Gear predictor-corrector method. Some of the mentioned communication sources can be modified to decrease the communication time. The time required for building the BDL probably is the one most affected by the increased number of processors. Since the search for the BDL data involves all the other processors an increase in the number of the processors directly affects the communication time. The required number of communications in the form of receive and send procedures is $2n_p(n_p - 1)$ where n_p is the number

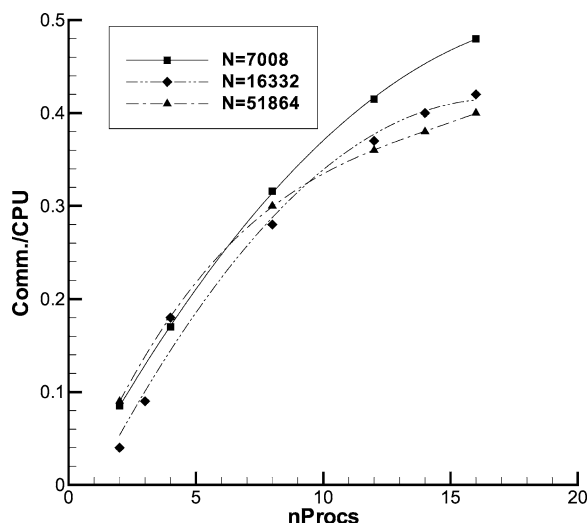


Fig. 11. The ratio of the communication time to the CPU time on each processor against the number of processors for three benchmark systems.

of processors in the virtual machine. So doubling the number of processors increases the communication time about four fold for building the BDL. So any technique that can reduce the communication effort for building the BDL can significantly improve the efficiency of the algorithm. However other techniques such as BNL (bonded neighbor list) that copies only the information of 3 neighbors from each direction entails the same communication workload as the data has to travel from one processor to all the rest to complete the required information. Overall, the branched version of our parallel algorithm still provides good speed up and efficiency about the same values that we obtained for the simulation of linear chains. So in terms of efficiency and speed up we do not see any major change. Our domain decomposition algorithm still proves to be more efficient than the replicated data method [20] used with PVM.

6. Conclusions

An algorithm for the simulation of branched molecules was described in this work. This algorithm is designed for parallel computations using a domain decomposition method. A message passing software handles the communication between the processors. Branching adds to the complexity of the algorithm and here we have devised techniques to handle these problems and still at the same time benefit from the speedup using the message passing method. We have put emphasis on how to handle intramolecular interactions for branched molecule with one level of branching and have provided means to extend the algorithm for molecules with higher level of branching. The algorithm proposed here is only one possibility that has proved to work effectively. We have used this algorithm to simulate and study the effect of branching on the rheological properties of alkane molecules. The results of this study is published in Ref. [21].

Acknowledgement

We gratefully acknowledge the support of this study by an Australian Research Council (ARC) grant. We wish also to thank for the generous time allocated to us on the computing facility of the Sydney Distributed Computing (SyDCom) Laboratory.

Appendix A

Listing of the program for calculating intermolecular interaction as described in Section 4.3. For a pair of atoms i and j that are located on the same chain all five interaction provisions are tested in this part. Expression CYCLE takes the computation out to the next pair of atoms.

```

!If the pair belong to the same chain and they are on the backbone.
  IF(((inBranch(atomId(j1))=0).AND.(inBranch(atomId(j2))=0)).OR.
  C  (((inBranch(atomId(j1))=0).AND.(inBranch(atomId(j2)).NE.0)).AND.
  C  (BranchSite (atomId(j2)).NE.0)).OR.(((inBranch(atomId(j2))=0).AND.
  C  (inBranch(atomId(j1)).NE.0)).AND.(BranchSite(atomId(j1)).NE.0)).OR.
  C  (((inBranch(atomId(j2)).NE.0).AND.(inBranch(atomId(j1)).NE.0)).AND.
  C  (BranchSite(atomId(j1)).NE.0).AND.(BranchSite(atomId(j2)).NE.0)))THEN
    IF(.NOT.((abs (atomId(j1)-atomId(j2)).gt.3)))CYCLE
  ELSEIF((inBranch(atomId(j1))=inBranch(atomId(j2))).AND.(BranchSite(atomId(j1))=0).AND.
  C  (BranchSite(atomId(j2))=0))THEN
    IF(.NOT.((abs (atomId(j1)-atomId(j2)).gt.3)))CYCLE
  ELSEIF((inBranch(atomId(j1))=inBranch(atomId(j2))).AND.((BranchSite(atomId(j1)).NE.0).OR.
  C  (BranchSite(atomId(j2)).NE.0)))THEN
!For pair of atoms on the same branch where one is the head of the branch
  IF(.NOT.((abs (BranchRank(atomId(j1))-BranchRank(atomId(j2))).gt.chain_LJ_limit)))CYCLE
  ELSEIF(((inBranch(atomId(j1))=0).AND.(inBranch(atomId(j2)).NE.0).AND.
  C  (BranchSite(atomId(j2))=0)).OR.((inBranch(atomId(j1)).NE.0).AND.
  C  (inBranch(atomId(j2)).NE.0).AND.(inBranch(atomId(j1)).NE.inBranch(atomId(j2))).AND.
  C  (BranchSite(atomId(j2))=0) )THEN
    IF(.NOT.((abs (atomId(j1)-BranchSiteId(atomId(j2))).gt.(3-BranchRank(atomId(j2))+1))))CYCLE
  ELSEIF( ((inBranch(atomId(j2))=0).AND.(inBranch(atomId(j1)).NE.0).AND.
  C  (BranchSite(atomId(j1))=0)).OR.((inBranch(atomId(j2)).NE.0).AND.
  C  (inBranch(atomId(j1)).NE.0).AND.(inBranch(atomId(j2)).NE.inBranch(atomId(j1))).AND.
  C  (BranchSite(atomId(j1))=0) )THEN
    IF(.NOT.((abs (atomId(j2)-BranchSiteId(atomId(j1))).gt.(3-BranchRank(atomId(j1))+1))))goto10
  ELSEIF((inBranch(atomId(j2)).NE.0).AND.(inBranch(atomId(j1)).NE.0).AND.
  C  (inBranch(atomId(j2)).NE.inBranch(atomId(j1))).AND.(BranchSite(atomId(j1))=0).AND.
  C  (BranchSite(atomId(j2))=0) )THEN
    IF(.NOT.((abs (BranchSiteId(atomId(j2))-BranchSiteId(atomId(j1))).gt.(3-BranchRank (atomId(j1))-
  C  BranchRank(atomId(j2))+2) )))CYCLE
  ENDIF

```

The listing below is for the calculation of stretching force in the bonds as described in Section 4.4.1. Parameters nMolecules, bondCounter and bondCounterBranch are number of the molecules, number of atoms on the backbone and number of atoms on the branch. The rest of the listing is self explanatory:

```

DO n=1, nMolecules
  IF(BDL(n,1,1)=0)cycle
  DO i=1,bondCounter(n)
    IF(BNL(n,i,2)=0)CYCLE
    DO j=1,bondCounterBranch(n,i)-1
      nn=BDL(n,i,j)

```

```

nn1=BDL(n,i,j+1)
!do the distance and force calculation for atoms nn and nn1
.....
ENDDO
ENDDO
ENDDO

```

The listing below is for the calculation of force from the angle potential described in Section 4.4.2.

```

! Stage one: sweep through the backbone.
DO n=1, nMolecules
  IF(BNL(n,1,1).eq.0)cycle
  IF(bondCounter(n).LT.3)cycle
  DO i=1,bondCounter(n)-2
    nn=BNL(n,i,1)
    nn1=BNL(n,i+1,1)
    nn2=BNL(n,i+2,1)
    !do angle force calculations for atoms nn, nn1 and nn2.
  ENDDO
ENDDO
! Stage two, go through the branches and angles between the backbone and branches.
DO n=1, nMolecules
  IF(BNL(n,1,1)==0)cycle
  DO i=1,bondCounter(n)
    IF(BNL(n,i,2)==0)THEN
      IF(i+1.LE.bondCounter(n))THEN
        IF(BNL(n,i+1,2)==0)CYCLE
      ELSE
        CYCLE
      ENDIF
    ENDIF
    !Go through the branch if it is long enough.
    IF((BNL(n,i,2).NE.0).AND.(bondCounterBranch(n,i).GE.3))THEN
      DO m=1,bondCounterBranch(n,i)-2
        nn=BNL(n,i,m)
        nn1=BNL(n,i,m+1)
        nn2=BNL(n,i,m+2)
        !do angle force calculations for atoms nn, nn1 and nn2.
      ENDDO
    ENDIF
    IF((BNL(n,i,2).NE.0).AND.((i+1).LE.bondCounter(n)))THEN
      nn=BNL(n,i,2)
      nn1=BNL(n,i,1)
      nn2=BNL(n,i+1,1)
    ENDIF
    IF(((i+1).LE.bondCounter(n)).AND.(BNL(n,i+1,2).NE.0))THEN
      nn=BNL(n,i,1)
      nn1=BNL(n,i+1,1)
    ENDIF
  ENDDO
ENDDO

```

```
nn2=BNL(n,i+1,2)
!do angle force calculations for atoms nn, nn1 and nn2.
ENDIF
ENDDO
ENDDO
```

References

- [1] D. Dowson, Thin films in tribology, in: Proceedings of the 19th Leeds–Lyon Symposium on Tribology, 1992, pp. 3–12.
- [2] I. Sendjarevic, A.J. McHugh, *Macromolecules* 33 (2000) 590–596.
- [3] P. Wood-Adams, *J. Rheology* 45 (2001) 203–210.
- [4] U.J. Jonsson, *Wear* 232 (1999) 185–191.
- [5] A. Thompson, M.O. Robbins, *Phys. Rev. A* 41 (1990) 6830–6837.
- [6] A. Jabbarzadeh, J.D. Atkinson, R.I. Tanner, *J. Non-New. Fluid Mech.* 69 (1997) 169–193.
- [7] A. Jabbarzadeh, J.D. Atkinson, R.I. Tanner, *J. Non-New. Fluid Mech.* 77 (1998) 53–78.
- [8] S.D. Gupta, H.D. Cochran, P.T. Cummings, *J. Chem. Phys.* 107 (1997) 10 316–10 326.
- [9] J.I. Siepmann, M.C. Martin, C.J. Mundyand, M.L. Klien, *Mol. Phys.* 90 (1997) 687–693.
- [10] B. Smit, S. Karaborni, I. Siepmann, *J. Chem. Phys.* 102 (1994) 2126–2140.
- [11] A. Jabbarzadeh, J.D. Atkinson, R.I. Tanner, *Comput. Phys. Comm.* 107 (1997) 123–136.
- [12] A. Jabbarzadeh, J.D. Atkinson, R.I. Tanner, *J. Chem. Phys.* 110 (1999) 2612–2620.
- [13] S.T. Cui, P.T. Cummings, H.D. Cochran, *Fluid Phase Equili.* 141 (1997) 45–61.
- [14] T.K. Xia, J. Ouyang, M.W. Ribarsky, U. Landman, *Phys. Rev. Lett.* 69 (1992) 1967–1970.
- [15] J.P. Ryckaert, A. Bellemans, *Chem. Phys. Lett.* 30 (1975) 123–125.
- [16] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, V. Sundream, *PVM: Parallel Virtual Machine. A Users Guide and Tutorial for Networked Parallel Computing*, MIT Press, Cambridge, MA, 1994.
- [17] W. Smit, *Comput. Phys. Comm.* 62 (1991) 229.
- [18] D.C. Rapaport, *The Art of Molecular Dynamics Simulation*, Cambridge University Press, 1995.
- [19] A.C. Raine, D. Fincham, W. Smith, *Comp. Phys. Comm.* 55 (1989) 13.
- [20] F. Muller-Plathe, W. Scott, W.F. van Gunstern, *Comp. Phys. Comm.* 84 (1994) 102.
- [21] A. Jabbarzadeh, J.D. Atkinson, R.I. Tanner, *Tribology Internat.* 35 (2002) 35–46.