Parallel Algorithms for MD-Simulations of Synthetic Polymers*

Bernd Jung

Editorial office "Macromolecular Chemistry and Physics"

Hegelstr. 45

D-55122 Mainz, Germany

Hans–Peter Lenhof Peter Müller Christine Rüb

Max-Planck-Institut für Informatik

D-66123 Saarbrücken, Germany

February 14, 1997

Abstract

Molecular dynamics simulation has become an important tool for testing and developing hypotheses about chemical and physical processes. Since the required amount of computing power is tremendous, there is a strong interest in parallel algorithms. We deal with efficient algorithms on MIMD computers for a special class of macromolecules, namely synthetic polymers, which play a very important role in industry. This makes it worthwhile to design fast parallel algorithms specifically for them. Contrary to existing parallel algorithms, our algorithms take the structure of synthetic polymers into account which allows faster simulation of their dynamics.

*This work is supported by the DFG–Schwerpunktprogramm "Effiziente Algorithmen für diskrete Probleme und ihre Anwendungen", grant LE 952/1–1

Contents

List of Figures

List of Tables

1 Introduction

With the advent of fast and affordable digital computers, a new tool for testing and developing hypotheses about chemical and physical processes became available: the technique of molecular dynamics simulation.

In a molecular dynamics (MD) simulation, the motions of all atoms of a given molecular system are simulated using classical mechanics. Given the atomic coordinates and velocities at time $t - \Delta t$ and t, the atoms' new positions at time $t + \Delta t$ are calculated by numerically integrating Newton's equation of motion (for more details, see Section 2).

The core of every MD simulation is the so-called *force field*, a set of analytical expressions that describes the potential energy in the molecular system. A great deal of work has been put into the development and improvement of such force fields for different kinds of molecules as their quality determines a simulation's usefulness. Therefore, MD simulations are not only used to predict properties that are not accessible by direct measurement but also to explain already known properties, thus improving knowledge and theory on the molecular and atomic level.

Today's main problem with MD simulations is the massive amount of computing power which is necessary to simulate molecular systems during a desired period of up to one second. As the above mentioned time step Δt is tiny (of the order of $10^{-15}s$) compared to the typical simulation periods, a tremendous number of work-intensive iterations has to be carried out. In many cases and especially when dealing with large molecules (thousands of atoms), this is far beyond the frontiers of today's sequential computer technology. For this reason, parallel algorithms for MD simulations have become one of the most active research fields in computational chemistry and physics.

We concentrate our work on parallel MD–simulations for a special class of molecules that play a major role in industry: synthetic polymers. Despite their importance, there are no parallel algorithms which have been especially adapted to them. Most simulations are done with algorithms which have been designed for biopolymers. Our approach makes better use of the structure of synthetic polymers which allows faster simulation of their dynamics.

The outline of this report is as follows. First, a short introduction to polymers and biopolymers is given and it will become clear that due to their different chemical and physical properties, different simulation algorithms should be used for these two types of macromolecules. Then, the basic ideas of molecular dynamics simulations are presented. In the following chapter, a sequential algorithm is described whose key ideas and techniques will also be employed in our parallel algorithm. Finally, some simulation results are given.

1.1 What is a polymer ?

In general a polymer is a substance whose single molecules are very big, i.e., containing hundreds and thousands of atoms, and consist of many basic units (the monomers) that are linked to form long chains. There are quite a few possibilities for categorizing polymers; a fundamental one is the distinction between *biopolymers* and *synthetic polymers*.

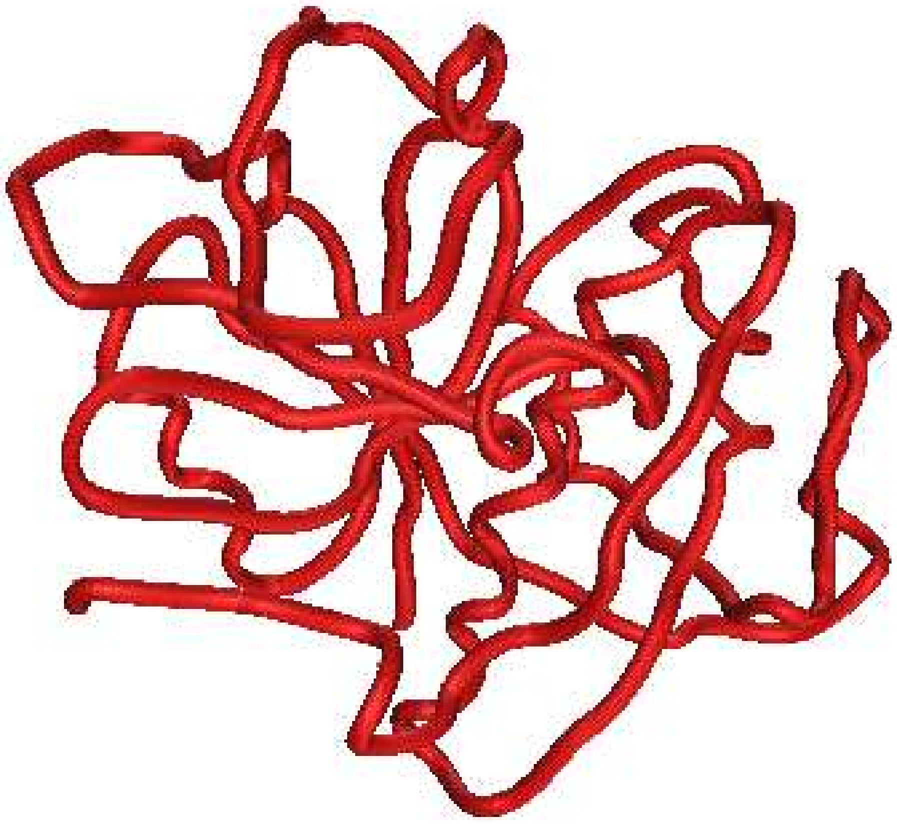

Figure 1: Backbone[1]representation of a protein.

Biopolymers are a class of naturally occuring molecules like proteins, DNA, RNA, polysaccharides or natural rubber. One molecule usually consists of many different types of monomers, e.g., in the case of proteins, these basic units are the amino acids. As proteins are the dominating biopolymers in publications,

[1]A backbone shows just the principal structure of a molecule by omitting details like single atoms, side groups, etc.

"biopolymer" is used synonymously with "protein" and we will also observe this convention. The typical 3D structure (the conformation) of a biopolymer is rather compact and a so-called "collapsed coil"(see Figure 1).

As its name already reveals, a synthetic polymer cannot be found in nature. It is produced in laboratories and chemical plants in a process called polymerization where monomers are prompted to link. Typically, a synthetic polymer consists of just a few types of monomers and often just one. Synthetic polymers are the base of all varieties of plastic (e.g., polyethylene, PVC, nylon, polyester) and therefore, many chemists simply call them "plastics". A single molecule of a synthetic polymer has a very different conformation from a biopolymer: it forms an "expanded random coil" (see Figure 2).

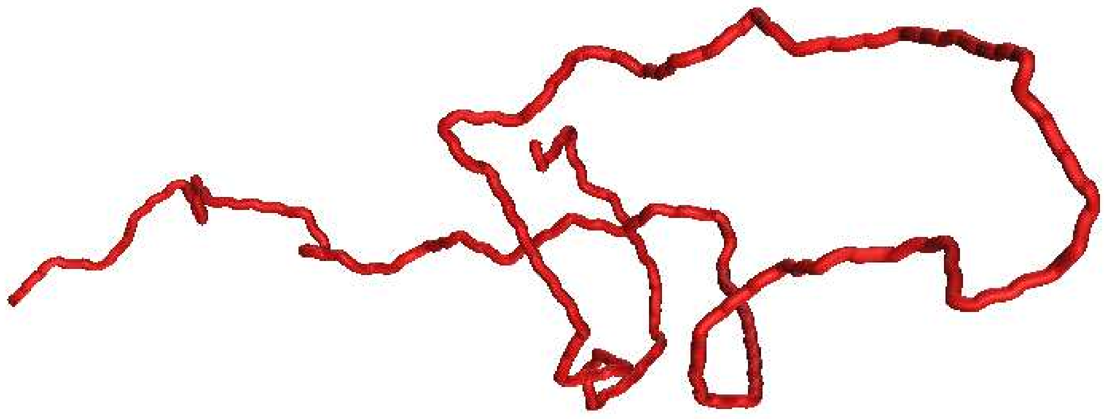

Figure 2: Backbone representation of a polymer

It should be obvious that these two kinds of polymers are very different with regard to their reaction and their chemical and physical properties so that no general all-purpose MD simulation algorithm that is efficient for both of them should be expected.

1.2 Parallel MD simulations for biopolymers

Many parallel algorithms for MD simulations for biopolymers have been developed and implemented (e.g., [BCL+92, BYT91, JP92, NHG+95, SSS92]), in most cases for distributed-memory architectures. Because of the compact conformation of biopolymers, nearly all algorithms use the following spatial decomposition method. The protein is placed in a small box which is decomposed into cubes (also called cells). These cubes are distributed among the processors and every processor is in charge of all atoms in its cubes and computes the interaction forces and new coordinates for these atoms. If an atom moves to a cube belonging to a different processor, responsibility for the atom passes to the new processor. Since proteins have a relatively fixed conformation (in comparison with synthetic polymers), only very few atoms move to another cube in an iteration step and the amount of data that must be exchanged between processors is small. Thus, the communication overhead caused by the parallelization is low, especially as communication takes place mainly between neighbouring processors. Furthermore, it is relatively easy to achieve a good load balancing among the processors due to the biopolymer's compactness, i.e., every cube will have approximately the same number of atoms.

1.3 Parallel MD simulations for synthetic polymers

The spatial decomposition method outlined above has also been applied to the simulation of synthetic polymers ([Fin90, SBR+92]). But unlike proteins, most synthetic polymers do not have a fixed conformation. They form expanded random coils and their trajectories are more or less three dimensional random walks. There is a lot of "empty space", i.e., space filled by other macromolecules or solvent molecules. Since simulating the dynamics or non-equilibrium properties can be done by simulating a single macromolecule of the polymer — the influence of the solvent and surrounding molecules can be modeled by random forces, frictional factors (Langevin dynamics) and mean–field interactions — many cubes in a spatial decomposition will be empty. This makes it much more difficult to achieve a good load balance between the processors. In addition, a synthetic polymer covers relatively long distances during the simulation compared to biopolymers whose dynamics are limited to some kind of vibration in a small area. Therefore, it does not seem to be efficient to divide the possible conformation area into cubes and allocate them to processors because a continual redistribution would be necessary as the polymer hastens away.

For these reasons, we decided to employ a different strategy. Instead of distributing the conformation space among the processors, we distribute the atoms. Naturally, this is somewhat dependent on the structure of the polymer (linear, combed or star shaped). As a beginning, we have implemented algorithms for the easier case of linear chains which will be explained in detail later on. These algorithms should also work for more complex polymer structures but less efficiently. In the future, we will concentrate on the modifications and enhancements that are necessary to improve the efficiency for them.

2 The basics of MD simulations

The goal of MD simulations is the investigation and analysis of structural and (thermo)dynamic properties of molecular systems. There are two main approaches to this goal.

The first one is based on *quantum mechanics* where approximations to Schrö/-din/-ger's equation are solved in order to determine electron density, energy and other molecular properties. Because of the enormous computational effort, this method is only feasible for a very small number of atoms and is not useful for most molecular systems.

Therefore, almost all MD simulations are based on *molecular mechanics*. The idea takes a somewhat coarse–grained view (compared to the quantum mechanical level) of the molecular system. Atoms are looked upon as balls, and bonds and other atom–atom interactions are modeled by analytical expressions. This makes it possible to use methods from the field of *classical mechanics*, especially the well–known Newton's law of motion, which are much easier and faster to compute than quantum mechanical equations; thus, bigger systems with several thousand atoms can be simulated.

2.1 Classical MD simulation

The motion and properties of a molecular system are simulated by computing the trajectories of each atom. This is done in an iterative way as follows.

The simulation period is divided into intervals of length Δt. Given the coordinates and velocities of all atoms at time $t - \Delta t$ and time t, the new positions at $t + \Delta t$ are calculated via numerical integration of Newton's Law:

$$m_i \cdot \vec{a}_i(t) = m_i \frac{\partial^2 \vec{q}_i(t)}{\partial t^2} = \vec{F}_i(t),$$

where

m_i	: mass of atom i,
$\vec{q}_i(t)$	: position of atom i at time t,
$\vec{F}_i(t)$	: force on atom i at time t,
$\vec{a}_i(t)$	: acceleration of atom i at time t.

Twice integrating with respect to t leads to the well–known equation of motion

$$\vec{q}_i(t) = \vec{u}_i \cdot t + \frac{1}{2}\vec{a}_i(t) \cdot t^2,$$

where $\vec{u}_i$: initial velocity of atom i, $\vec{a}_i(t)$: acceleration of atom i at time t.

Taylor–expansion yields

$$\vec{q_i}(t+\Delta t) = \vec{q_i}(t) + \vec{v_i}(t)\Delta t + \frac{1}{2}\vec{a_i}(t)\Delta t^2 + O(\Delta t^3)$$
$$\vec{q_i}(t-\Delta t) = \vec{q_i}(t) - \vec{v_i}(t)\Delta t + \frac{1}{2}\vec{a_i}(t)\Delta t^2 - O(\Delta t^3).$$

These equations can be combined to

$$\vec{q_i}(t+\Delta t) = 2\vec{q_i}(t) - \vec{q_i}(t-\Delta t) + \frac{\vec{F_i}(t)}{m_i}\Delta t^2(+O(\Delta t^4)),$$

which is called *Verlet–Algorithm.* It is one of the most widespread methods used in MD simulations for calculating the new positions.

Computing the new positions is straightforward, but what about the necessary forces $\vec{F_i}(t)$? They are the result of the energetic interactions between all atoms in the system at time t and can be obtained from a so–called *force field* whose calculation is the gist of every step in an MD simulation.

A *force field* is a sum of analytical expressions that try to approximate the potential energy in the molecular system. Its name stems from the fact that the gradient of the force field represents the forces currently acting on the atoms. The components of a force field for classical MD simulations contain the potentials for :

- covalent bonds
- valence angles
- dihedral angles
- non–bonded pairs of atoms (van der Waals and electrostatic interactions).

The force field that we use in our simulations only takes short–range interactions into account (i.e., no electrostatic potentials; these will be added in the future) and looks like this:

$$V = \sum_b K_b(r_b^2 - r_{b,0}^2)^2 + \sum_v K_v(r_{1-3}^2 - r_{1-3,0}^2)^2 +$$
$$\sum_{1-4,i<j}\left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^6}\right) + \sum_{nb,i<j}\left(\frac{A_{ij}}{r_{ij}^{12}} - \frac{B_{ij}}{r_{ij}^6}\right).$$

r_b is the current length of the bonds (b) and r_{1-3} is the current 1–3 distance[2] in the valence angles (v). The subscript 0 denotes the values with minimal potential

[2]A valence angle is defined by three atoms. 1–3 distance is the distance between the first and the third atom.

energy. r_{ij} is the distance between atoms i and j. A_{ij} and B_{ij} are the Lennard–Jones $12-6$ parameters for van der Waals potentials and are used not only to calculate the non–bonded interactions (nb), but also to express the dihedral angle interactions (1–4). Evaluation of forces by the derivatives of these expressions is much faster than using the standard ones. In [Jun93a], the force field given above was derived together with equations that allow to calculate the necessary parameters from those of the AMBER force field (see [WKN+86, WKC+84]).

When the force field for time t has been evaluated, the impact on each atom, i.e., the current force that pushes it to a new position, is calculated and applied according to the equations given above.

So, the basic structure of a classical MD simulation is as follows:

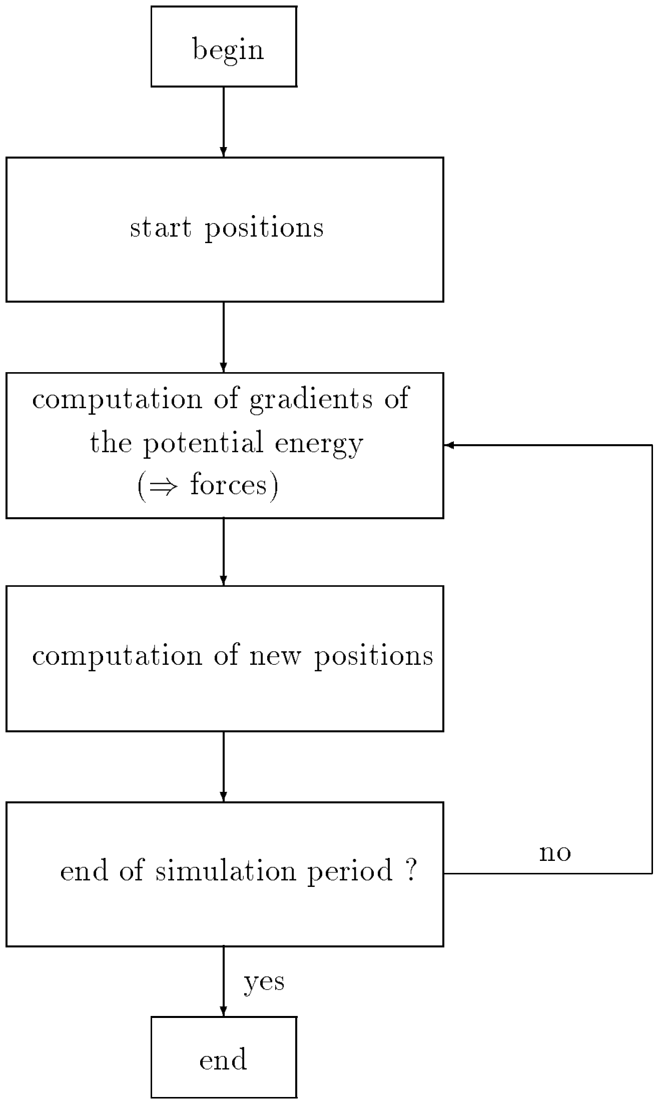
begin
start positions
computation of gradients of the potential energy (⇒ forces)
computation of new positions
end of simulation period ?
no
yes
end

Figure 3: The basic structure of a classical MD simulation

2.2 Langevin–Dynamics simulation

As molecules do not exist in a vacuum, it is necessary to include the atoms of the environment (e.g., a liquid) and of other macromolecules in the simulation. Unfortunately, this increases the number of particles and consequently the complexity of the computation considerably. However, in certain cases (e.g., when simulating non–equilibrium properties like diffusion) it is possible to emulate the influence of these additional atoms by using the *Langevin–equation*

$$m_i \cdot \vec{a_i}(t) = -\gamma_i \cdot \vec{v_i}(t) + \vec{R_i}(t),$$

where γ_i denotes a friction constant for atom i and $\vec{R_i}(t)$ is a Gauss–distributed random force acting on atom i at time t. When using the Verlet–Algorithm, this leads to the modified formula ([PBS88])

$$\vec{q_i}(t+\Delta t) = \vec{q_i}(t) + [\vec{q_i}(t) - \vec{q_i}(t-\Delta t)](1-\gamma_i \Delta t) + \frac{\vec{F_i}(t) + \vec{R_i}(t)}{m_i}\Delta t^2.$$

The resulting MD simulation is called *Langevin–Dynamics*. Its advantage is the fact that the numerous atoms of the surrounding area can be omitted because their influence is taken into account by

- a friction component (represented by the γ_i factors) that simulates the friction between molecule and solvent, and
- a stochastic force component $\vec{R_i}$ (with mean value $< R_i >= 0$ and variance $< R_i^2 >= 2\gamma_i k_B T/\Delta t$, $\quad k_B$: Boltzmann's constant, T : absolute temperature) which models the pushes that the atoms may be given by surrounding atoms.

In this way, a realistic simulation is feasible with considerably fewer atoms than otherwise necessary.

2.3 Acceleration techniques

Because of the enormous computational requirement for MD simulations, many methods have been devised to accelerate the simulation without losing too much precision ([VG91]). Usually, these techniques are employed together but we are going to present them here separately for reasons of clarity.

2.3.1 Cutoff radius

In principle, every atom has an influence on every other atom in the system that is to be simulated, so we face an n–body problem with $O(n^2)$ atom pairs. Luckily, the intensity of an interaction between two atoms decreases with distance. Therefore, it is often possible to neglect all atoms that are beyond a given cut–off radius by setting their energy/force contribution to zero. This method works fine for fast–decreasing potentials like the Lennard–Jones potential which approximates van der Waals interactions.

Note that a cutoff–radius just reduces the amount of work for computing the potentials; there are still $\Theta(n^2)$ atom pairs to be looked at in every iteration in order to find out whether they are within the cutoff radius.

2.3.2 Neighbour–lists

A typical time–step in MD simulations is of the order of $10^{-15}s$. During this period, the atoms cannot move much. So, if an atom was within the cutoff–radius for some atom at time t, in most cases, it will not have left the radius at time $t + \Delta t$. This finding led to the idea of neighbour lists: every atom has a list naming the atoms that are within the cutoff–radius at some time. This list is used in k subsequent time steps to evaluate the potentials between the atoms and its neighbours. Then the neighbour lists are updated. The expensive determination of neighbouring atoms within the cutoff–radius is therefore reduced by a factor of k.

2.3.3 Multiple–time–step method

The length of the integration time step Δt is limited by the highest frequency (ν_{max}) motions occuring in the molecular system: $\Delta t \ll \nu_{max}^{-1} = \tau$. Three frequency ranges can be distinguished:

- high–frequency bond–stretching forces with an approximate time τ of about 10^{-14}s,
- low–frequency Coulomb–forces with $\tau \approx 10^{-12}$s,
- the remaining intermediate frequency forces with $\tau \approx 4 \cdot 10^{-14}$s.

The contribution of these three kinds of forces can be evaluated by using three different time steps without much loss of accuracy.

2.3.4 Linked cells

When the number of atoms n exceeds a few hundred, determining neighbourship becomes the most time–consuming part of the simulation if this is done by looking at all $\frac{n(n-1)}{2}$ atom pairs.

Therefore, in the linked–cells method, the conformation space is subdivided into cells with appropriate edge lengths. All atoms that are in the same cell are kept in a linked list. Determining neighbours of an atom i can then be limited to the cell (or list, respectively) where i is located, and the bordering cells.

Because of symmetry (Newton's third axiom *actio* = *reactio*), the search can even be limited to the cells that are lexicographically bigger than the current cell (see Figure 4).

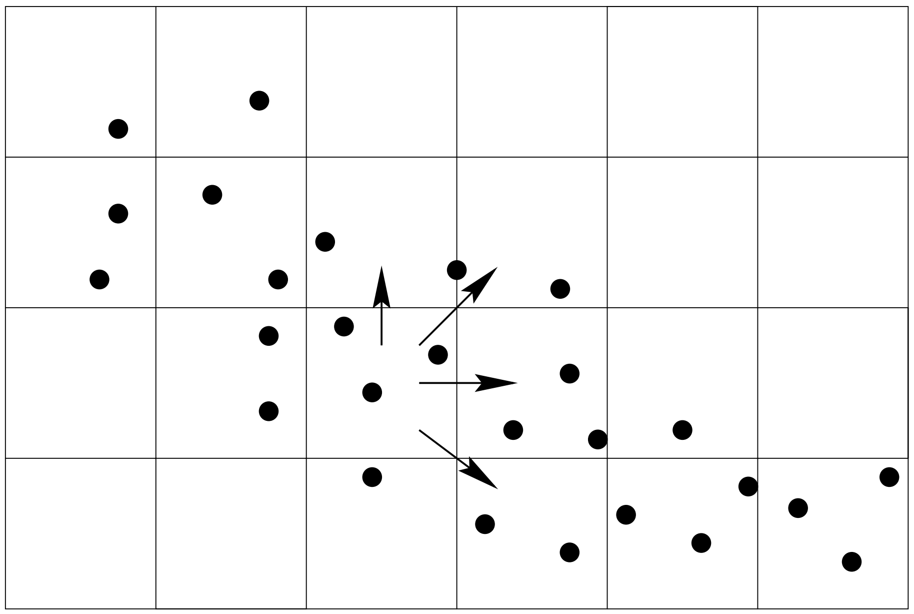

Figure 4: linked cell method in two dimensions

3 A sequential MD simulation algorithm for synthetic polymers

3.1 Simulation model

As the dynamic behaviour of synthetic polymers can be studied by simulating a single macromolecule, we use the Verlet–formula for Langevin–dynamics. Our algorithms use the force field presented in Section 2, with parameters and constants recalculated from the AMBER–package ([WKN+86, WKC+84]). The employed force field expressions for bonds, valence angles and dihedral angles (developed by [Jun93a]) allow a much faster computation than the usual expressions because no "expensive" operations like square root or trigonometric functions are necessary.

The time–step Δt for the simulation is $0.5 \cdot 10^{-15} s$. Calculation of the force field is done with a multiple–time–step scheme: covalent bonds are evaluated in every time–step whereas interactions from valence angles and dihedral angles are considered every second and fourth time–step, respectively. Van der Waals forces are calculated every fourth time step with a cutoff radius of 4 Å. For long–range van der Waals interactions, only the repelling part of the 12–6 Lennard–Jones potential is applied in order to avoid collapse of the coiled macromolecule. Thus, a good solvent is simulated.

Due to the long simulation runs, a great deal of "good" random numbers must be created. We have chosen the "ran1" algorithm [PFT+88], which is quite time–consuming, but yields random numbers of acceptable quality.

At present, electrostatic interactions are not considered. This will be implemented in the near future. Furthermore, all algorithms presented in this work are specially suited to linear macromolecules. More complex structures will lead to reduced efficiency.

3.2 The algorithm

As explained in the previous section, there are two main parts of the MD simulation during each iteration: calculation of the forces and calculation of new atom positions due to these forces. From an algorithmic view, the latter is trivial, so we will focus on the computation of the force field.

The force field approximates the potential energy (in the molecular system) which results from all interactions that exist between atom pairs. We can classify atom pairs into two groups:

- atom pairs with interactions of type covalent bond, valence angle or dihedral angle, and
- atom pairs with interactions of type van der Waals or electrostatic.

The treatment of atom pairs of the first group is straightforward. The reason for this is that their defining interaction (covalent bond, valence and dihedral angle) is "static" with respect to the simulation: the interacting atoms are defined by the structure of the molecule and cannot change in the course of time. Hence, it is possible to perform a preprocessing step by constructing lists for covalent bonds, valence angles, and dihedral angles, where each list element contains the indices of the atoms that interact. In every step of the simulation — when the appropriate force field component must be evaluated — we run through the appropriate list, read the current coordinates of the atoms involved, and compute the forces.

Atom pairs of the second group are more difficult to deal with. As already mentioned in Section 2, only atoms that are within a certain cutoff radius are of importance for the computation. Because of the atoms' motions, relevant atom pairs come into being or cease to exist depending on the current distance between them. Therefore, no preprocessing is possible and an expensive neighbourhood search must be performed in each iteration step of the simulation. Luckily, there are atom pairs whose distance is always smaller than the cutoff radius because they are "neighbours" in the polymer chain. For these pairs, a preprocessing step similar to the one for the first group of atom pairs can be done. We call all atom pairs that can be stored in one of these lists *static pairs*. Note that not all atoms within the cutoff radius are considered for these static pairs (e.g., in the case of polyethylene, atoms covered by the first group and atoms that are farther away than 2 CH_2 units do not contribute to static pairs).

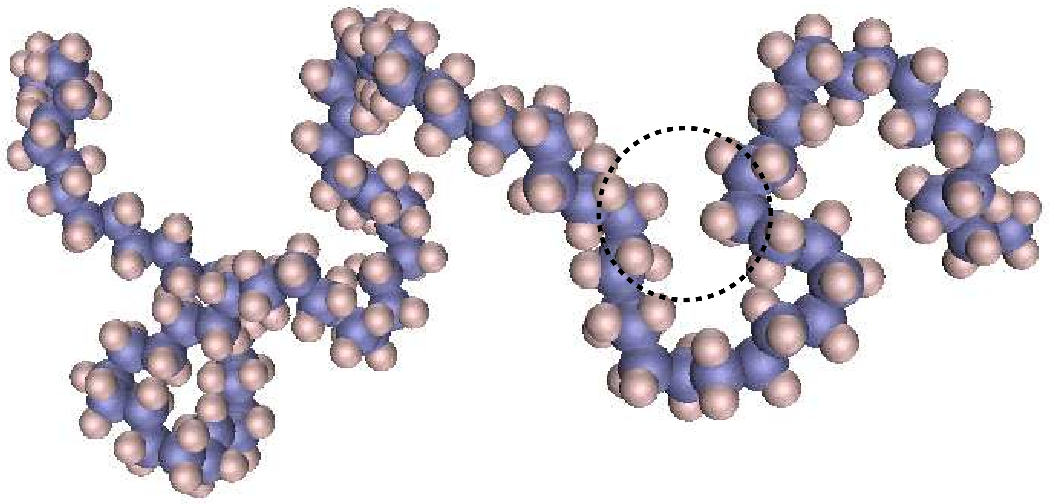

Figure 5: Dynamic pairs arise from bends in the chain

Since the polymer chain builds bends or loops during the simulation (see Figure 5), some atoms will be close enough for van der Waals interactions (i.e., within the cutoff radius) for a certain time. Because their relationship evolves dynamically and changes continually after a few iterations, we call these pairs of atoms *dynamic pairs*. Obviously, no preprocessing step is possible, so whenever van der Waals interactions have to be computed, we must determine all current dynamic pairs by using a "fixed–radius–all-nearest–neighbours–algorithm".

We use the following standard technique to do this: a mesh of cubes is laid over the molecule and its conformation area. Each cube is identified by a vector (xyz). For each non–empty cube, two lists are maintained:

- the first list contains the (indices of the) atoms that currently reside in the cube

- the second list contains pointers to all neighbouring cubes which are not empty (at most 26).

Of course, the cube size must be chosen properly (edge length $\geq$ cutoff radius) so that relevant neighbouring atoms will be in one of the 26 surrounding cubes and not in cubes farther away.

All non–empty cubes are held in a hash–table. When potentials from dynamic pairs are to be evaluated, this table is run through and for every cube the following is done:
All atoms belonging to this cube are tested with all atoms of neighbouring non–empty cubes (since the diameter of the cubes is small, there are only very few atoms in each cube and the quadratic running time here is negligible). If they are close enough, their contribution to the force field is calculated. As the interaction between two atoms is symmetric (apart from the sign), it is sufficient to consider only neighbouring cubes that are lexicographically bigger than the current cube.

After the computation of the new positions of the atoms, i.e., after the Verlet–step, some atoms will have changed their cube. Therefore, in all iterations that precede an iteration where Van der Waals interactions due to dynamic pairs must be considered (i.e., in every fourth iteration) a check for every atom is made to see whether the atom has moved on to a different cube. In this instance, the cubes concerned are updated accordingly.

4 A parallel MD simulation algorithm

4.1 Machine model

We developed our parallel algorithms with the following machine model in mind. Given is a collection of independent processors, each with its own local memory, which are linked together by some interconnection network. The processors communicate by sending/receiving messages and are numbered from 0 to $p-1$, where p is the number of processors.

This model of a parallel machine is called *MIMD (= multiple instruction multiple data)* because every processor executes its own program in its local memory and all processors are completely independent of one another apart from explicit communication via message–passing. Many existing parallel computers, especially those used for simulation, are covered by this model, e.g., Connection Machine 5 (Thinking Machines), iPSC/860 (Intel), Paragon XP/S (Intel), Cray T3D/E (Cray Research).

As parallel computing technology is still very expensive, most installations have only a moderate number of processors. Even systems with relatively many processors are often configured in such a way that the costly resource “processing unit” is partitioned into subsets so that the average user can access just a small number of processors. Against this background the parallel algorithm presented in the following sections has been developed for a moderate number of processors (up to 16).

4.2 The algorithm

One main problem in parallelizing an algorithm is the discrepancy between local memory access times and remote memory access times. The latter is much more "expensive" and must therefore be kept at a minimum.

With this in mind, we have to tackle the two different kinds of interactions in the MD simulation that have been presented in Section 3: static and dynamic atom pairs.

The static atom pairs are defined by the molecular structure and do not change over time whereas dynamic pairs evolve and vanish during the simulation. Not surprisingly, it is easier to handle the static pairs in parallel and therefore, we will show how to deal with interactions due to static pairs first. This already gives an algorithm which is useful for specific applications, e.g., the simulation of Θ–conditions [Jun89c, DSA91, SMB94]. After this, we show how to add the interactions occuring because of dynamic pairs.

4.2.1 Interactions between static pairs

In order to keep communication due to static pairs minimal, it is necessary to divide the atoms among the processors in such a way that the number of atom pairs with atoms belonging to different processors is small. As already mentioned, we restrict the class of polymers to be simulated to those with chain–like structure (e.g., unbranched polyethylene).

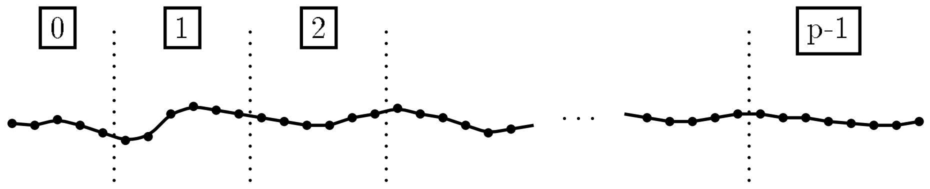

Figure 6: The chain is divided into disjoint segments.

In this case, we can use the following straightforward approach. Let p be the number of processors and n the number of atoms. We subdivide the polymer chain into p segments of roughly size n/p (see Figure 6).

These segments are distributed among the processors. Each processor is responsible for all atoms in its segment, i.e., in each iteration it computes the forces acting on these atoms and their resulting new coordinates. By declaring one end of the polymer chain "left end", we impose an order on the segments. Every segment has a left and/or right neighbour segment. Accordingly, a processor has a left and/or right neighbour. (Note that this has nothing to do with physical neighbours in the interconnection network.)

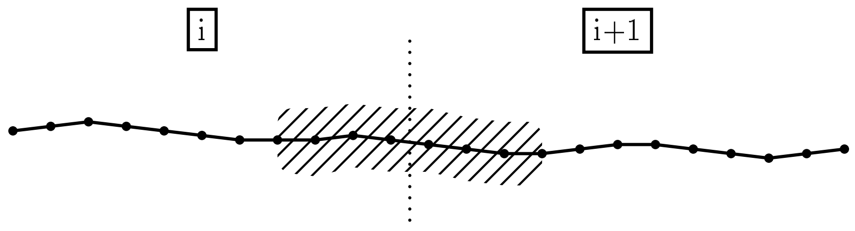

Figure 7: Border atoms of two neighbouring processors

For each processor, we distinguish between "inner" atoms and "border" atoms. Border atoms are located close enough to the border of a processor's segment such that they interact with atoms in the neighbouring segment (which is assigned to a different processor); all other atoms are inner atoms (see Figure 7).

The parts of a segment containing only border atoms are called left and right border region. The size of such a border region varies with the kind of polymer that is being simulated. For polyethylene, at most four CH_2 groups, that is 12 atoms, form such a border region. We assume that the segment size n/p is large enough so that left and right border regions of a segment do not overlap. (In our implementation, this is checked at the beginning and if the condition is not met, only some of the available processors will be used.)

The algorithm works as follows. One processor acts as a "master": it reads the input (start positions, chemical parameters that define the molecule, etc.), distributes the atoms among the processors and collects the output (positions, velocities and other data). Except for this, it does no coordination of the processors during execution.

In a preprocessing step, neighbouring processors exchange information about their border atoms, for instance, the atom type. During the actual simulation, at the beginning of every iteration each processor sends the coordinates of its border atoms to the corresponding neighbouring processors. This is done by non–blocking sends and receives : after initiating the exchange of data, each processor proceeds by computing the interactions for its inner atoms where no information from the outside is needed. Meanwhile, the required data for the border atoms pour in and, when the inner atoms are done, the calculation for the border atoms can begin. On many modern parallel computers, communication and local computation can take place simultaneously, thus allowing the communication time to be hidden.

When all interactions from inner and border atoms have been computed, each processor calculates the forces and resulting new positions of the atoms in its segment.

After a certain number of iterations (that depends on the purpose of the simulation), the master collects output data: each processor sends its data (e.g., atom coordinates) to the master processor. If necessary (or desired), the master computes global information like the potential energy or other simulation parameters.

If we cut the input molecule into equally sized segments, this approach will not lead to a perfect load balance between the processors as the processors responsible for the leftmost and rightmost segment of the chain possess fewer border atoms. Thus, they have to send and receive less data and the amount of local computation is smaller because their leftmost or rightmost atoms have fewer neighbours. Finally, the master processor has to do more work because it collects the output. To improve the running time of the algorithm it is therefore better to cut the chain into unequally sized segments. The best choice for the lengths of the segments depends on the particular computer used. In our implementation, the master processor is also responsible for the leftmost segment. For the simulation of a polyethylene molecule on an iPSC/860, we assigned approximately 10% more atoms to the leftmost and rightmost segment than to the other segments.

Table 1: Static pairs only, iPSC/860

nodes	time in s	speedup efficiency	speedup factor
1	11175	100.0	1.0
2	5625	99.3	1.99
4	2912	95.9	3.84
8	1499	93.2	7.45
16	780	89.5	14.33
32	431	81.0	25.93

Tables 1 and 2 show running times for the simulation of a polyethylene chain consisting of 1000 CH_2 units. The simulation has been carried out on an Intel iPSC/860 and on an Intel Paragon XP/S. The results shown are for 40000 iteration steps, which corresponds to $2 \cdot 10^{-11}$s in reality. The tables show the running times, the achieved speedup when compared with the running time on a single processor (speedup factor), and the speedup as percentage of the maximum possible (speedup efficiency). As one can see, the speedup grows with increasing p, and the speedup factor ranges from an excellent 1.99 and 1.98 with 2 processors to a still good 25.93 and 27.38 with 32 processors.

As can be seen by the running times, our algorithm achieves a good load balance between the processors. The (inevitable) drop in the speedup efficiency with increasing number of processors stems from two sources. One reason is that

Table 2: Static pairs only, Paragon XP/S

nodes	time in s	speedup efficiency	speedup factor
1	6625	100.0	1.0
2	3347	99.0	1.98
4	1714	96.3	3.87
8	878	94.3	7.55
16	443	93.5	14.95
32	242	85.6	27.38

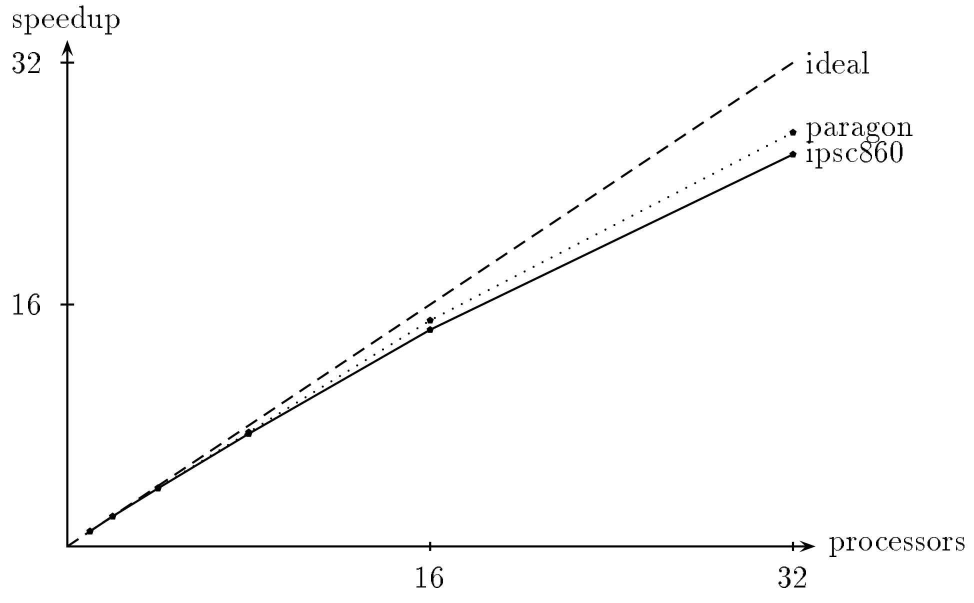

Figure 8: Speedup for static pairs

part of the calculations is carried out twice: the sequential algorithm makes heavy use of symmetries, i.e., the force acting between any two atoms is calculated only once. In the parallel algorithm, this is no longer the case: the forces acting between two border atoms are calculated twice, namely by the two processors responsible for the two atoms. That means that the total work is increased by a term that is linear in the number p of processors: The sequential running time t_s is approximately $c \cdot n$, where n is the number of atoms and c is a constant. The parallel work w_p is approximately $c \cdot n + c' \cdot (p-1)$, where c' is a constant that depends on the number of border atoms per processor. Since the speedup efficiency is given by $t_s/w_p \cdot 100$, it will drop with increasing p. The second reason for the decrease in efficiency is the communication overhead: initiating the send or receive of a message and preparing/evaluating messages requires a certain amount of time.

Similar results can be expected for other chain types as long as the number of static atom pairs with atoms allocated to different processors remains small.

4.2.2 Interactions between dynamic pairs

In the previous section we described how to compute the interactions for static pairs; now we show how to include the interactions between dynamic pairs.

Interactions because of dynamic pairs confront us with the problem of determining the atoms that form these dynamic pairs at a given moment. In Section 3, we described a sequential algorithm that lays a mesh of cubes over the molecule and its conformation area in order to do the dynamic neighbours search. Unfortunately, dynamic pairs can evolve between any two sections of the polymer chain. If these sections belong to the piece of the chain that has been allocated to one processor, then no communication is necessary and this processor can use the sequential method for detecting local dynamic pairs.

But there may be also pairs with the atoms allocated to different processors. In order to find them, the positions of all atoms in the molecule are necessary but each processors knows only the piece of the chain that it has been given. As we want to keep communication as low as possible, the trivial method where each processors sends the coordinates of all its atoms to every other processor is obviously not recommendable. Therefore, we devised the following master–slave approach that takes use of the efficient sequential algorithm:

A mesh of cubes is laid over the molecule and its conformation area. Each processor maintains a data structure similar to the sequential version for all atoms belonging to its part of the chain. Additionally, a master processor maintains a "global" data structure. With the help of this data structure, the master processor determines which processors have to exchange the coordinates of atoms due to the formation of loops or bends. To be able to do this, the master processor must know which cubes contain atoms and to which processors they belong. This is achieved as follows.

Dynamic neighbours need only be considered after each 4th step; thus, after every 4th step, each processor determines which of its atoms have moved to a different box. This information is sent to the master processor which updates the global cube data structure. After this, the master processor assembles for each processor a list of pairs (i, j), where i is the number of a processor and j denotes a cube. If a pair (i, j) is contained in the list of processor k, this means that processor k has to send the coordinates of all of its atoms in box j to processor i. The size of these lists is kept to a minimum by applying the following rules: No pair is contained more than once in a list. Interactions between static pairs are not considered, i.e., the master processor checks, for each pair of non-empty boxes, whether all atoms stem from two neighbouring border regions. In this case the pair is not included. The list of processor k will not contain pairs of the form $(k, -)$. When the master has prepared all the lists, it sends them to the processors.

After receiving the lists, every processor k sends for each pair (i, j) the coordinates of the atoms that are contained in box j to processor i. After the receipt of all relevant data, each processor proceeds as in the sequential algorithm.

We chose this approach for the following reasons. As long as we can distribute the work equally among the processors, it is most important that we keep the overall work as low as possible (assuming there are no waiting times). If we distribute the global cube data structure (or similar global information) among several processors, more work will be carried out and we will increase the amount of communication.

Tables 3 and 4 show running times of the dynamic version for the simulation of a polyethylene chain consisting of 1000 CH_2 units on an iPSC/860 and Paragon XP/S. Again, the simulation has been carried out for 40000 iteration steps.

Table 3: Static and dynamic pairs, iPSC/860

nodes	time in s	speedup efficiency	speedup factor
1	13038	100.0	1.0
2	6801	95.9	1.92
4	3644	89.4	3.58
8	1933	84.3	6.74
16	1064	76.6	12.25
32	—[3]	—	—

Table 4: Static and dynamic pairs, Paragon XP/S

nodes	time in s	speedup efficiency	speedup factor
1	8273	100.0	1.0
2	4384	94.4	1.89
4	2283	90.6	3.62
8	1210	85.5	6.84
16	664	77.9	12.46
32	503	51.4	16.44

[3]not available

The speedup efficiency varies from 95.9% and 94.4% with 2 processors to 76.6% and 77.9% for 16 processors. There are several reasons for the decrease in efficiency compared with the static version.

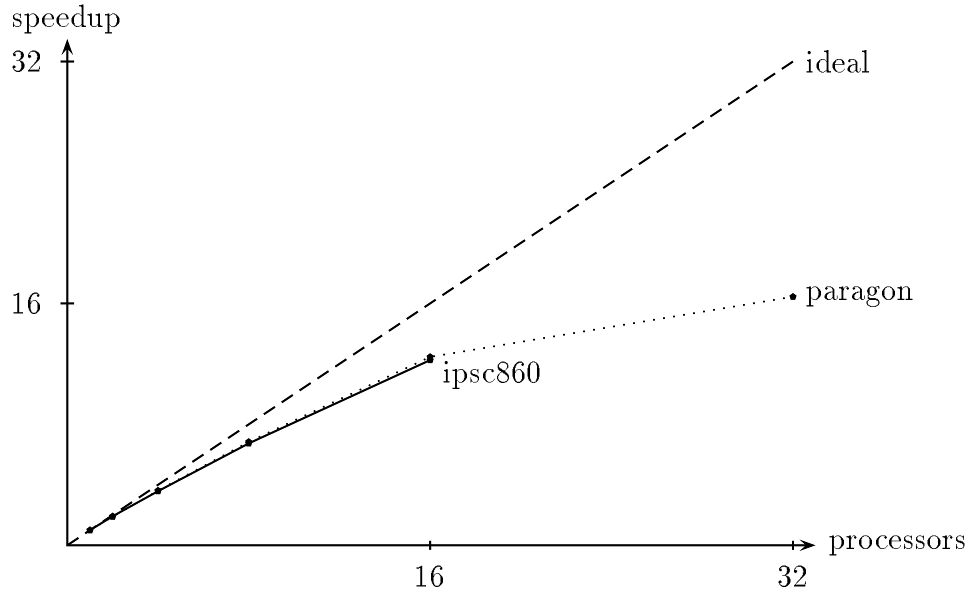

Figure 9: Speedup for static and dynamic pairs

Because of the dynamic interactions, an expensive neighbourhood search must be done. This is the part of the simulation which is hardest to parallelize and which increases the communication and computation overhead considerably. Apart from the additional work the load balance between processors gets worse (compared to static pairs only) because in general only a few processors — if any — have to exchange information (=atom coordinates) due to dynamic neighbours. This kind of imbalance is aggravated by imbalances between the four steps that constitute one "superstep": right now, in one of the steps (where dynamic pairs have to be evaluated according to the simulation model) the master processor has more work to perform than the other processors and in all other steps, it has less work than the others. All these factors inevitably lead to a smaller but still quite acceptable speedup for moderate numbers of processors (up to 16). In the future we will investigate the power of dynamic load balancing techniques to improve efficiency and speedup.

5 Some simulation results

In order to check the correctness of the implementation of our parallel MD-simulation algorithm, the dynamics of polyethylene chains of different lengths were simulated. During these simulations, some static and dynamic properties of the chains were monitored.

In all figures, the results of the parallel version of the algorithm are compared with those of the sequential version. It can be seen that the agreement of the parallel simulations with the sequential ones for Θ–conditions is excellent and the scaling law $< r^2 > \sim n$ is fulfilled.

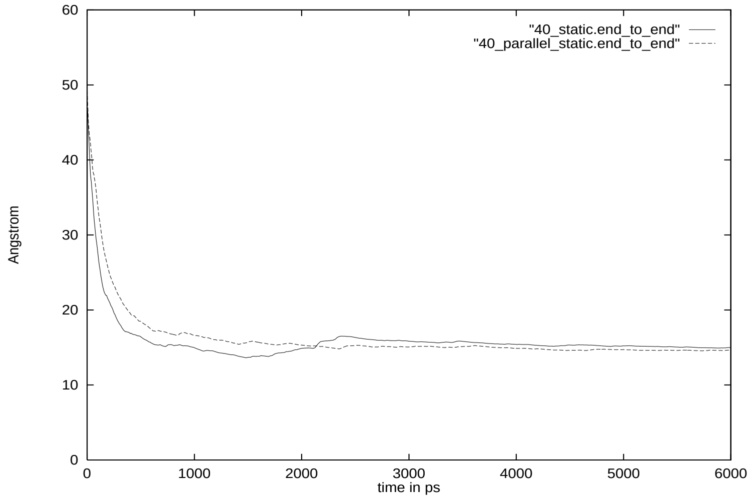

Figure 10: End–to–end distance, 40 C–atoms, Θ–condition

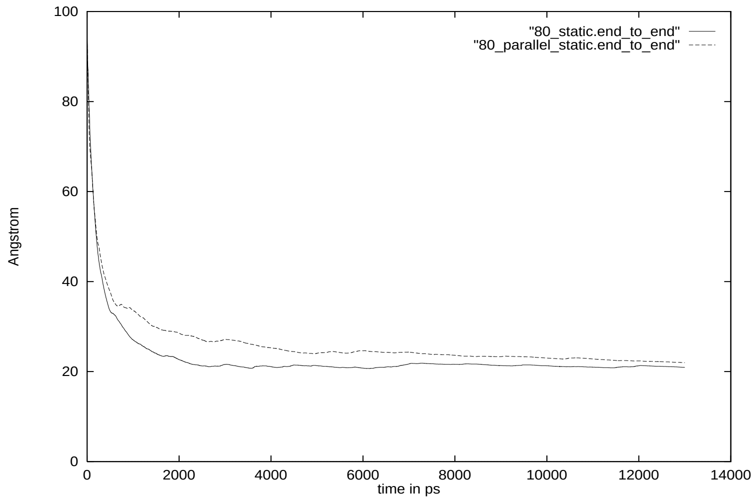

Figure 11: End–to–end distance, 80 C–atoms, Θ–condition

Figures 10 and 11 show the root of the cumulative mean square end–to–end distance ($\sqrt{< r^2 >}$) of polyethylene chains with $n = 40$ and $n = 80$ C–atoms, respectively, under Θ–conditions, i.e., only static pairs are considered.

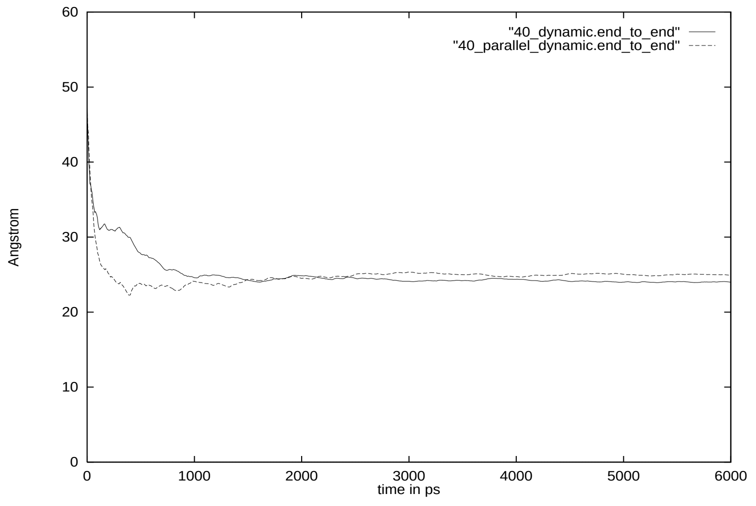

Figure 12: End–to–end distance, 40 C–atoms, good–solvent condition

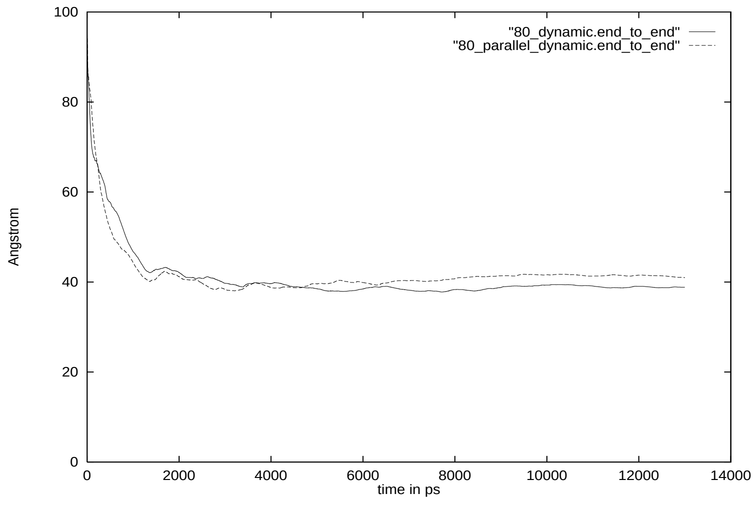

Figure 13: End–to–end distance, 80 C–atoms, good–solvent condition

Figures 12 and 13 show the same comparison for good–solvent conditions, i.e., considering "dynamic pairs" with repulsive van der Waals interactions. As to be expected, the coils are expanded relative to Θ–conditions.

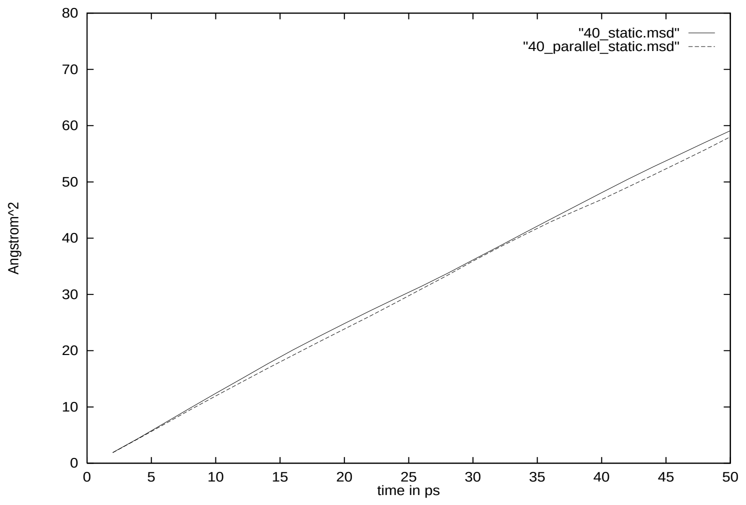

Figure 14: Mean square displacement, 40 C–atoms, Θ–condition

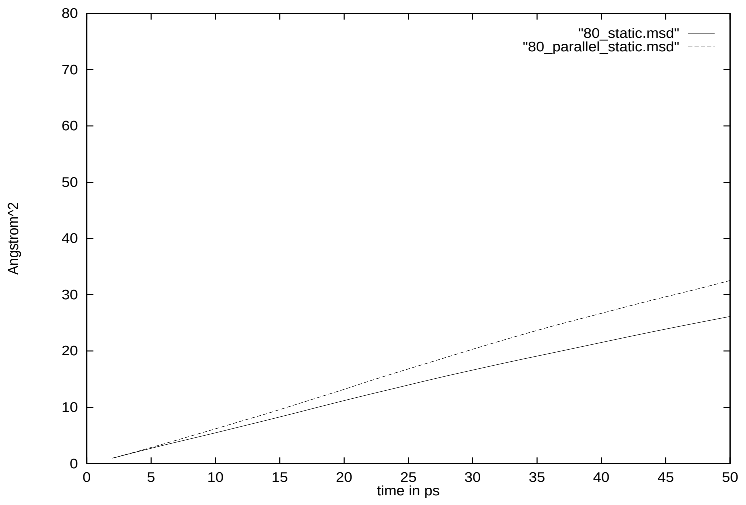

Figure 15: Mean square displacement, 80 C–atoms, Θ–condition

Figures 14 and 15 show the mean square displacement of the center of mass plotted versus time for chains under Θ–conditions. These data are extracted from simulations starting from conformations after full relaxation of the end–to–end distance.

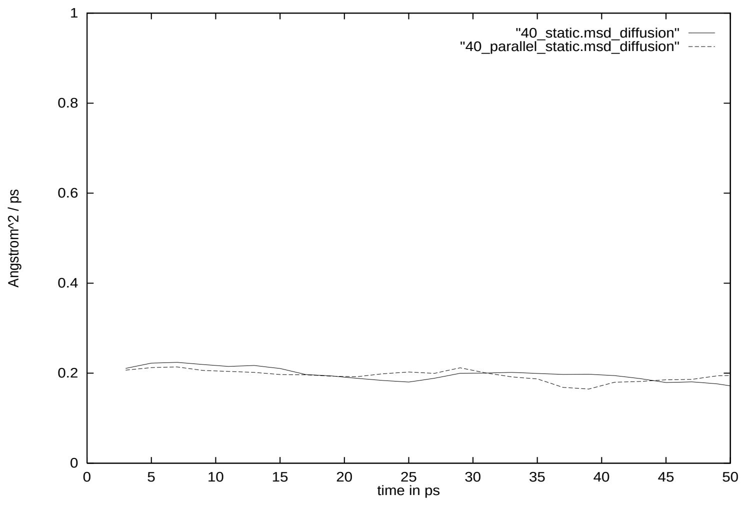

Figure 16: Diffusion coefficient, 40 C–atoms, Θ–condition

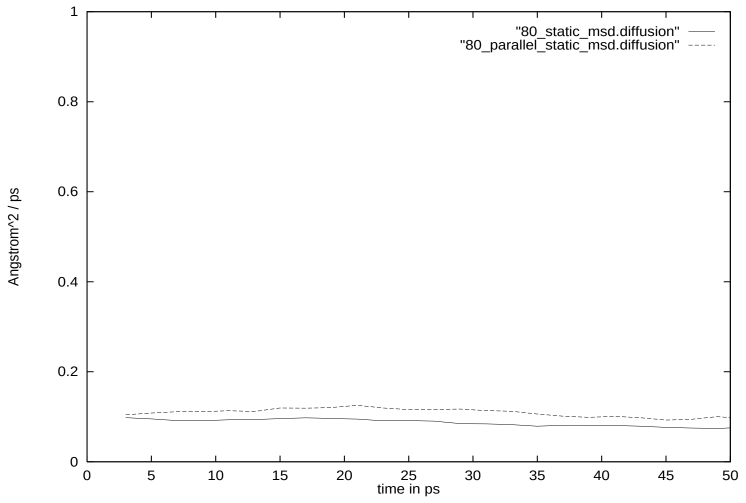

Figure 17: Diffusion coefficient, 80 C–atoms, Θ–condition

Figures 16 and 17 show the corresponding first derivatives divided by 6, i.e., the instantaneous diffusion coefficients D. Also for this dynamic property the agreement between sequential and parallel simulations is good. Furthermore, the expected power law $D \sim \frac{1}{n}$ for Rouse conditions (i.e., without hydrodynamic interactions) is approximately fulfilled.

Figures 18 to 21 show the same plots for simulations under good–solvent conditions. The agreement between sequential and parallel simulations in this case is still acceptable, but worse than under Θ–conditions.

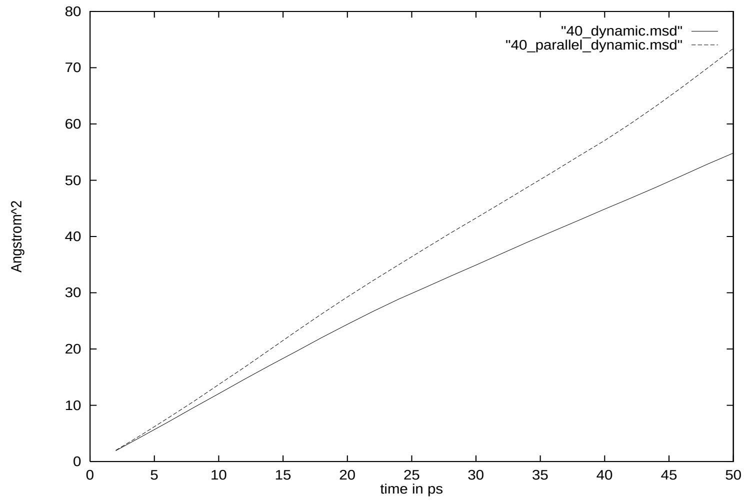

Figure 18: Mean square displacement, 40 C–atoms, good–solvent condition

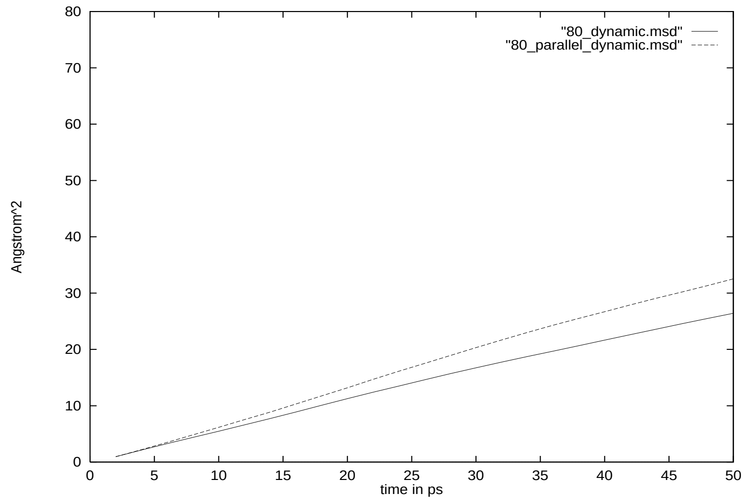

Figure 19: Mean square displacement, 80 C–atoms, good–solvent condition

The reason is that here the dynamic neighbours are considered. These dynamic interactions are indeed hydrodynamic interactions because the interacting sections of a chain mutually influence their motions. Therefore, the average of D

should be taken over the whole space of conformations, while in fact the simulation time was too short for this. However, the power law of $D \sim \frac{1}{\sqrt{n}}$ for Zimm conditions (i.e., with hydrodynamic interactions) is at least roughly fulfilled.

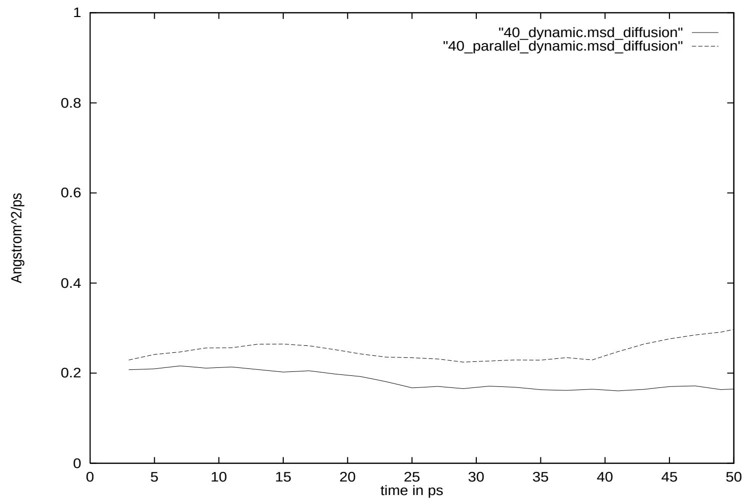

Figure 20: Diffusion coefficient, 40 C–atoms, good-solvent condition

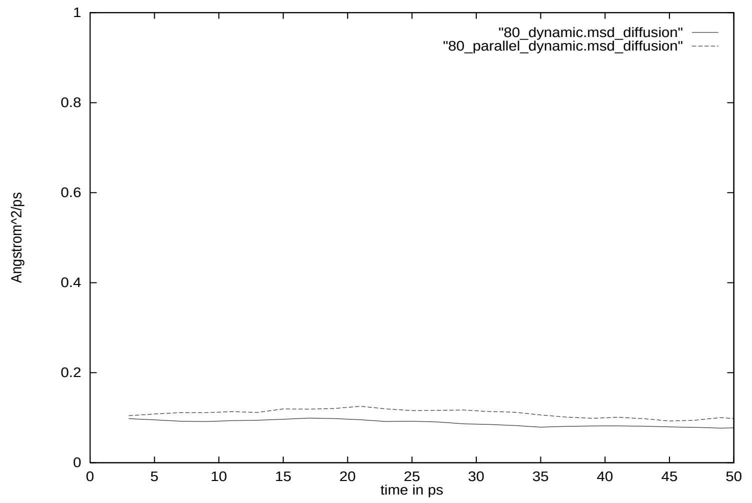

Figure 21: Diffusion coefficient, 80 C–atoms, good–solvent condition

In summary, we can say that the results of the simulations using the parallel algorithm compare favourably with the theoretical expectations and with the results of the sequential calculations.

6 Future work

There are two main goals for the future.

First, we want to test (and modify where appropriate) the algorithms for a broader range of synthetic polymers. At the moment, only simulations for polymers with a linear chain have been considered in the implementations; but as some synthetic polymers appear as branched or comb–like chains, it would be nice to have a modified version of our algorithms that can deal with them, too.

Second, we will enhance the existing algorithms by integrating a dynamic load balancing scheme. Due to dynamic pairs, some processors have more work to do (data exchange and calculations) compared to the beginning of the simulation when the initial subdivision into segments was done. For this reason, the work is no longer balanced and this slows down the simulation. We will try to develop methods that detect such imbalances and eliminate them by redistributing the atoms of the molecule.

The presented algorithms perform well for a moderate number of processors (up to 16). But as more and more parallel computers with large numbers of processors come into use, we must also develop suitable algorithms for these systems. Obviously, our "master–slave" approach is not scalable to large numbers of processors and therefore no longer very efficient because of increasing communication overhead.

References

[AT89] M. P. Allen, D. J. Tildesley : "Computer Simulation of Liquids". Oxford University Press, 1989.

[BCL+92] J. A. Board, J. W. Causey, J. F. Leathrum, A. Windemuth, K. Schulten: "Accelerated molecular dynamics simulation with the parallel fast multipole algorithm". Chemical Physics Letters (2 Oct. 1992) vol. 198, no. 1–2, pp. 89–94

[BYT91] C. L. Brooks, W. S. Young, D. J. Tobias: "Molecular simulations on supercomputers". International Journal of Supercomputer Applications (Winter 1991) vol. 5, no. 4, pp. 98–112.

[DSA91] M. Depner, B. L. Schürmann und F. Auriemma: "Investigation of a poly(oxethylene) chain by a molecular dynamics simulation in an aqueous solution and by Langevin dynamics simulation". Mol. Phys. **74**, (1991), pp. 715–733.

[Fin90] D. Fincham: "Parallel Computers and the simulation of solids and liquids". in "Computer Modelling of Fluids, Polymers and Solids". C. R.

A. Catlow, S. C. Parker, M. P. Allen (Eds.), NATO ASI Series, Ser. C, vol. 293. Kluwer Academic Publishers, Dordrecht (1990), pp. 269–288.

[HFP+94] R. Haberlandt, S. Fritzsche, G. Peinel, K. Heinzinger : "Molekulardynamik". Vieweg, 1994.

[Hai92] J. M. Haile : "Molecular Dynamics Simulation". Wiley, 1992.

[JP92] J. F. Janak, P. C. Pattnaik: "Protein calculations on parallel processors". Journal of Computational Chemistry (Nov. 1992) vol. 13, no. 9, pp. 1098–1102

[Jun89c] B. Jung: "Simulation der Kettenkonformation von Polymeren mit Hilfe der Konzepte der Molekulardynamik-Rechnungen", Dissertation, Johannes-Gutenberg-Universität, Mainz, 1989.

[Jun93a] B. Jung: "Fast force field expressions for computer simulations". Macromol. Chem., Theory Simul. **2**, (1993), pp. 673–684.

[NHG+95] M. Nelson, W. Humprey, A. Gursory, A. Dalke, L. Kale, R. Skeel, K. Schulten, R. Kufrin : "MD–Scope: A Visual Computing Environment for Structural Biology". Computational Physics Communications 91, 1995, pp. 111–134.

[Oet96] H. C. Öttinger: "Stochastic processes in polymeric fluids". Springer Verlag, Berlin 1996.

[PBS88] R.W. Pastor, B.R. Brooks, A. Szabo: "An analysis of the accuracy of Langevin and molecular dynamics agorithms", Mol. Phys. **65**, (1988), pp. 1409–1419.

[PFT+88] W. H. Press, B. P. Flannery, S. A. Teukolsky, W. T. Vetterling, "Numerical recipes in C". Cambridge University Press, Cambridge 1988.

[SMB94] A. Sariban, T. Mosell, J. Brickmann: "Monte Carlo simulation of the theta-conditions of poly(methylene): temperature dependence of the long-range screening factor", Macromol. Theory Sim., **3**, (1994), pp. 963–977.

[SBR+92] M. A. Smith, Y. Bar-Yam, Y. Rabin, B. Ostrovsky, C. H. Benett, N. Margolus und T. Toffoli: "Parallel-processing simulation of polymers". Comput. Polym. Sci. **2**, (1992), pp. 165–171.

[SSS92] H. Schreiber, O. Steinhauser, P. Schuster: "Parallel molecular dynamics of biomolecules". Parallel Computing (May 1992) vol. 18, no. 5, pp. 557-573.

[VG91] W. F. van Gunsteren: “Computer simulation of biomolecular systems: Overview of time-saving techniques”. Advances in Biomedical Simulations, AIP Conference Proceedings No. 239, American Institute of Physics, (1991), pp. 131-146.

[WKN+86] S. J. Weiner, P. A. Kollman, D. T. Nguyen, D. A. Case. *An All Atom Force Field for Simulations of Proteins and Nucleic Acids*, Journal of Computational Chemistry, **7**, (1986) pp. 230–252.

[WKC+84] S. J. Weiner, K. A. Kollman. *A New Force Field for Molecular Mechanical Simulation of Nucleic Acids and Proteins*, Journal of American Chemical Society **106**, (1984) pp. 765–784.