Parallel Simulation of Polymers on the Cray T3E

M. Pütz, A. Kolb and B. Dünweg

Max Planck Institute for Polymer Research
Ackermannweg 10
D-55128 Mainz, Germany

Abstract. In this article we present a way of simulating polymer systems on the mesoscopic scale using massively parallel computers. We use a continuum bead-spring model and molecular dynamics with statistical noise and friction to propagate our systems in time.
For simple flexible polymer melts and networks our model contains only pair interactions. In this case spatial decomposition of the simulation box onto several processing units is the ideal way of parallelizing the problem. We describe the algorithm we use and the data structures involved in considerable detail. The communication overhead is quite small, giving rise to an almost perfect scaling of the algorithm up to 256 Cray T3E processing elements, the only requirement being that our dense systems exceed a particle number of roughly 400 particles per processor.
Furthermore we have developed a hybrid approach using both spatial and force decomposition for more complicated interactions along the backbone of the polymer chains. This approach allows us to study the behavior of semi-flexible polymers that have bending (three point) and rotational (four point) interactions. This hybrid algorithm proves to be very efficient: It scales very well with the number of processors, and in comparison with the simpler pair interaction algorithm it shows only a 15% - 20% drop in performance when used to simulate flexible polymers.

1 Introduction

Polymer systems pose a big challenge to computational physics. Typically, they exhibit a large variety of length and time scales which span over many orders of magnitude. For example, in a polymer melt the typical oscillation time of a C-C bond is of the order of picoseconds, while the time needed for a chain to diffuse its own size (or to relax its conformation) grows very strongly with molecular weight, and can easily reach up to the regime of micro- or even milliseconds. Therefore, it is obviously hopeless to run a Molecular Dynamics (MD) simulation of long-chain polymer systems which realistically models these C-C oscillations, and still attempts to reach the diffusive regime, which is necessary if the simulation is to provide a sample of configurations which can be used for

averaging in the sense of equilibrium statistical physics.
Fortunately, however, the many time scales are directly related to corresponding length scales: High frequencies / short times correspond to short length scales, while the longer time scales are related to larger length scales - it simply takes more time to relax the structure of a larger object. In many cases this relation actually shows up in the form of simple power laws which directly relate the relaxation time to the corresponding length scale. To mention again the case of a polymer melt: According to the so-called reptation model [1], which describes the experimental data for the viscoelastic behavior reasonably well, the relaxation time τ is proportional to the third power of the degree of polymerization, N. Likewise, the typical chain conformations in the melt are described by random walks, and hence the typical size of the coil, R, is proportional to $N^{1/2}$. Combining these laws, one sees that $\tau \propto R^6$, i. e. an extremely strong increase of relaxation time with typical size.
The mesoscopic approach to polymer simulations is then to take advantage of these scaling properties, and look at the system at a length and time scale where the microscopic details have averaged out. The complicated many-body interaction between all the atoms is replaced by a simplified model which contains a considerably smaller number of degrees of freedom. More precisely, several chemical repeat units are combined to a single effective bead; this is the so-called "coarse-graining" procedure. If the number of original monomers is chosen large enough, the chain can be viewed as fully flexible on the scale of the coarse-grained model. Hence, the effective interactions which are used in the model system have to take into account only a few fundamental properties of polymers: Connectivity, i. e. the chain structure as such, excluded volume, i. e. the impossibility of two monomers sitting on top of each other, non-crossability (which is particularly important for the chain motion), and flexibility, which is usually modeled via spherically symmetric interactions. For details of the standard model which is used for many purposes in the theory group at the MPI for Polymer Research, see the next section.
Such models are much more appropriate for studying phenomena on the larger length and time scales. Of course, unless one knows how the parameters of the coarse-grained model (interaction constants, etc.) are related to the underlying chemistry, the relation to the latter is lost. Establishing a reliable relation between these two scales is a highly non-trivial task, and is currently developing into a full branch of research in its own right. However, there are also quite a number of phenomena which are not well understood, but are universal in the sense that they occur independently of the details of the underlying microscopic chemistry. For these questions it is justified to simply disregard the chemical detail, and instead study the system on the mesoscopic scale right at the outset. In practice, this means that as many parameters as possible are simply set to unity. For example, it is observed experimentally that a stretched polymer network exhibits an anisotropic scattering pattern, the so-called "butterfly effect". It has

been debated if this is a result of thermal fluctuations or rather of the intrinsic disorder which is a result of random crosslinking [2]. To clarify this question, we have run (and are currently still running) large polymer networks of up to 2×10^6 particles. Very large systems are needed, because the observed phenomena occur for rather large wavelengths. For systems like these, we have developed an efficient parallel algorithm which exploits the physical nature of the problem: Since the interactions are short-ranged, and the density fluctuations are small, domain decomposition is the method of choice, since only communication to (logically) adjacent processors is required, and the program is automatically rather well load-balanced. Very good performance has been obtained on Cray T3Es, which seems to be mainly a result of the excellent communication hardware of this architecture.
The remainder of this article is organized as follows: In Sec. 2 we define the model and the basic algorithm, whose scalar optimization is outlined in Sec. 3. Sec. 4 describes the basic parallelization strategy, while Sec. 5 outlines the generalization to the case of additional bending and torsional potentials along the chain, where we found a hybrid scheme (i. e. a mixture of spatial and force decomposition) more useful. Sec. 6 concludes with some benchmark data obtained on a Cray T3E.

2 Model and Algorithm

2.1 A Mesoscopic Polymer Model

The conformations of polymers which are much longer than their persistence length can be well described by simple bead-spring models on the mesoscopic scale. The model we employ has frequently been used in previous simulations for melts, networks and single chains [3]. Thus, many of its properties are well known, which makes it relatively easy to choose suitable parameters for a given problem. Since many details of the implementation depend on the exact definition of this model, we shall present it here for completeness.
We model a polymer chain via several monomers with mass m which are connected by finitely extendable non-linear elastic (FENE) springs. All monomers repell each other due to a repulsive Lennard-Jones interaction. The reason that the springs are not harmonic is mainly historic and not of fundamental importance. We want to make use of the results of previous simulations, therefore we keep the FENE springs. If we have M chains with N_i monomers each, the Hamiltonian for the model is

$$H\left(\{\mathbf{p}_{i,j}, \mathbf{r}_{i,j}\}\right) = \sum_{i=1,j=1}^{M,N_i} \frac{\mathbf{p}_{i,j}^2}{2m_{i,j}} + \sum_{i=1,j=1}^{M,N_i-1} V_{FENE}\left(|\mathbf{r}_{i,j} - \mathbf{r}_{i,j+1}|\right) \qquad (1)$$

$$+ \sum_{i=1,j=1}^{M,N_i} \sum_{k=1,l=1}^{M,N_k} V_{LJ}\left(|\mathbf{r}_{i,j} - \mathbf{r}_{k,l}|\right)$$

where i,k label chains, j,l label monomers and

$$V_{FENE}(r) = \frac{\alpha}{2}\frac{R^2}{\sigma^2}\ln\left(1 - \frac{r^2}{R^2}\right) \quad \text{for} \quad r < R, \tag{2}$$

$$V_{LJ}(r) = 4\epsilon\left[\left(\frac{\sigma}{r}\right)^{12} - \left(\frac{\sigma}{r}\right)^{6} + \frac{1}{4}\right] \quad \text{for} \quad r < r_c = 2^{\frac{1}{6}}\sigma. \tag{3}$$

V_{FENE} and V_{LJ} are zero for other values of r. One sets $\epsilon = 1$ and $\sigma = 1$ which fixes the energy and length scales. This leaves us essentially three parameters: the mass parameter m, which can usually also be set to one if one does not include monomers of different mass, the strength α of the FENE springs and their extensibility R. The latter controls the relative importance of the excluded volume of the monomers. For dense melts and networks at a typical monomer density $\rho = 0.85$ we choose $\alpha = 30$ and $R = 1.5$. This choice serves a two-fold goal: First, the frequency of oscillation between two monomers coupled by a spring (and, of course, excluded volume interaction [see Fig. 1]) is approximately equal to the frequency of a monomer within the effective potential well caused by its surrounding monomers due to excluded volume interactions only. Therefore, we have only one natural high frequency cutoff in the system which is good if one employs a single time step integration scheme. Secondly, two different chains or two separated parts of the same chain have an effective energy barrier of about 70ϵ to cross through each other, and thus the topological constraint that two chains may not cross each other is effectively met for all reasonable temperatures (kT is typically one or of order one).

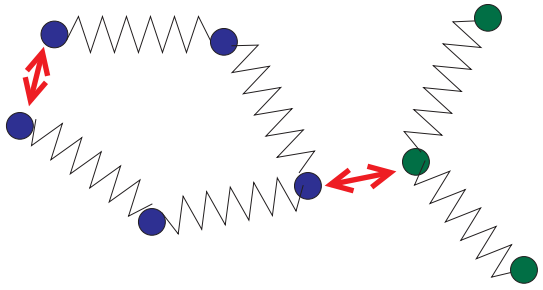

Figure 1: Bead-spring model: Monomers along a chain are connected with anharmonic springs, excluded volume (arrows) is modeled by Lennard-Jones repulsion.

2.2 Method of Integration

We use the standard velocity Verlet integration scheme to propagate our systems in time. In order to keep it stable at bigger time steps and to simulate a canonical ensemble instead of a microcanonical one, we further introduce a stochastic force $\mathbf{W}_{i,j}(t)$ on each monomer and friction term $\Gamma\dot{\mathbf{r}}_{i,j}$ with a friction coefficient Γ. Further one has to impose the following restrictions to the first and second moment of $\mathbf{W}_{i,j}(t)$:

$$\begin{aligned} \langle \mathbf{W}_{i,j}(t) \rangle &= \mathbf{0} \\ \langle \mathbf{W}_{i,j}(t) \cdot \mathbf{W}_{k,l}(t') \rangle &= 6kT\Gamma\delta(t-t')\delta_{i,k}\delta_{j,l} \end{aligned} \tag{4}$$

The latter couples $|\mathbf{W}|$, Γ and T according to the fluctuation-dissipation theorem. For the generation of the stochastic forces we use uniformly distributed random numbers in the range of $[-\Delta, \Delta]^3$ where $\Delta = \sqrt{\frac{24kT\Gamma m}{h}}$ and h is the time step of integration.
The velocity Verlet integrator then takes the following form:

$$\mathbf{r}(t+h) = \mathbf{r}(t) + h\mathbf{v}(t)\left(1 - \frac{h\Gamma}{2m}\right) + \frac{h^2}{2m}\mathbf{F}(t) \tag{5}$$

$$\mathbf{v}(t+h) = \left[\mathbf{v}(t)\left(1 - \frac{h\Gamma}{2m}\right) + \frac{h^2}{2m}\left(\mathbf{F}(t) + \mathbf{F}(t+h)\right)\right]\left(1 + \frac{h\Gamma}{2m}\right)^{-1} \tag{6}$$

$\mathbf{F}$ includes both potential and stochastic forces. Typically the parameters we use will vary around $T = 1.0$ and $\Gamma = 0.5$. The friction induced by the heat bath is roughly equal to $\frac{1}{40}$ of the effective friction between monomers.

3 Optimization of the Linked Cell Algorithm

Clearly the double sum in (1) is the most time consuming part of the force calculation. This is even more so since the particles' coordinates are not restricted to a small part of space. Therefore one has to check each pair of particles whether they are within interaction range or not. This brute force approach clearly scales with the square of the number N_{tot} of particles. Therefore it is only efficient for a relatively small number of particles up to $N_{tot} \approx 500$.
For larger systems one usually reduces the effort to linear order in N_{tot} using the *linked cell* scheme [4,5]. The main idea behind this is to divide the simulation box into small cells with a diameter equal to the interaction radius r_{ia} (see the coarse grid in Fig. 2). Then one sorts all particles into these cells according to their coordinates. For each cell one has to keep a pointer list of the particles it contains. To find all possible interacting pairs one has to go through all cells and each particle within them and scan only neighboring cells for interacting particles. This loop is illustrated by the following C code:

```
for(cell = 0;cell < lastCell;cell++)
  for(i = 0;i < particlesInCell[cell];i++) {
     iPointer = particleList[cell][i];
     for(j = i+1;i < particlesInCell[cell];j++) {
       jPointer = particleList[cell][j];
       CalcInteraction(iPointer,jPointer);
     }
     for(nCell = firstNeighCell[cell];nCell <= lastNeighCell;nCell++)
       for(j = 0;i < particlesInCell[nCell];j++) {
          jPointer = particleList[cell][j];
          CalcInteraction(i,j);
       }
  }
```

The *linked cell* algorithm reduces the search effort to the order

$$\frac{27}{2}\left(\rho r_{ia}^3\right)^2 \frac{N_{tot}}{\rho r_{ia}^3} = \frac{27}{2}\left(\rho r_{ia}^3\right) N_{tot} \tag{7}$$

where ρ is the particle density. The last factor $\frac{N_{tot}}{\rho r_{ia}^3}$ gives the number of cells, each of these has $3^3 = 27$ neighbors on a 3d lattice and contains ρr_{ia}^3 particles in average. The prefactor of $\frac{1}{2}$ accounts for the fact that two neighboring cells have to be checked only once during the search exploiting the actio equals reactio principle for pair forces. As a result the algorithm scales only linearly with N_{tot}, however, with a proportionality factor which can actually be quite large depending on the density and range of interactions.

If one compares the interaction sphere around one particular particle and compares it to the volume that is actually scanned around it by this algorithm one finds that only a fraction $\frac{4\pi}{81} \approx \frac{1}{6}$ of the particles really interact. To avoid these wasted distance calculations one should approximate the interaction sphere more closely. One way to do it would be to make the cells smaller and simultanously increase the number of neighbor cells (next nearest etc.) which one has to scan. However this method has the severe drawback of increasing the loop iterations dramatically while most of the cells will be empty. Thus, the loop overhead will eventually eat up the benefit of the saved distance calculations. Our own tests showed that this is already the case if one divides the cell size by a factor of two.

However, Everaers and Kremer [6] have shown how to exploit the fact that less particles occupy the cells. They devised a method they called *neighbor cell assignment* to achieve that only a single particle or none sits in one cell. This opened the opportunity to simplify the data structure which made their algorithm fully vectorizable and reduced the loop overhead significantly (two loops in the ordinary linked cell algorithm can be saved). The drawback of this algorithm is that it is relatively complicated and difficult to tune. Furthermore its superiority is limited on scalar architectures.

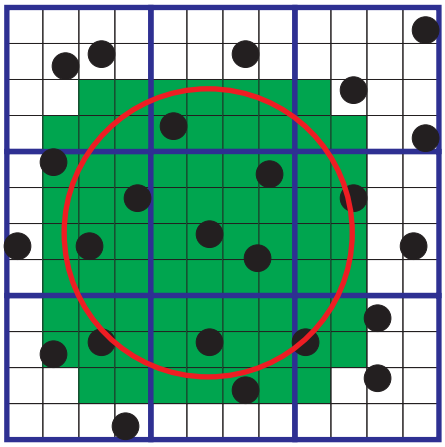

Figure 2: Neighbor cells (thick lines) with minicells (thin lines). The circle shows the interaction radius around one particle and how it is approximated by the minicells (shaded).

Instead we decided to try a different method of reducing the wasted volume keeping the loop overhead constant. We divide the cells again into n_{mini}^3 minicells and set up a table for each pair of neighbor cells and each pair of minicells therein which contains the boolean information whether two minicells in neighboring cells can possibly interact (i. e. are within r_{ia} of each other). Then we supplement the cell list which contains all particles within a cell with the additional information of the minicell in which each particle sits. This is a very simple calculation and can be done during the step when all particles are stored within this list. Before we do the actual distance calculation of two particles we first take a look at the minicell table to find out if their minicells are within interaction range. So we have exchanged the costly distance calculation involving 6 memory loads and 9 FLOPS by one memory load and a simple integer comparison for most of the potential interaction pairs.
Our experiences show us that the actual gain of this method depends strongly on non-universal factors such as memory speed (for table access) and the details of the lookup-table implementation. On a computer with 64 bit integers (such as the T3E) a choice of $n_{mini} = 4$ seems optimal since then the minicell interaction table can be stored in a compressed bit array minimizing memory loads and cache misses. Also the particle pointers in the cell list can store the minicell index in the unused upper bits of the pointers further reducing memory loads. This method of reducing memory loads by storing related pieces of information which are frequently accessed together in different bits of a single integer variable is a very good optimization technique on scalar computers. This

is particularly true if the memory caches are too small for the data fields to fit into them. Bit coding, of course, increases the number of instructions to extract a specific piece of information, however these bit manipulation instructions are usually very fast compared to the latency times of the main memory.
If we now look at the ratio of the amount of computing time spent on force calculations to the amount of time spent searching for interactions the result is still tantalizing: only 25% of the time are real work instructions, the rest is consumed almost completely by search loop overhead. This problem can however be tackled by the standard method of using a so-called Verlet table [4,5], which stores all pairs of particles with distance $r_{ia} + s$. The length s, which is also called *skin*, represents a safety shell around the actual interaction sphere. When one has to calculate the forces for the next time step one does not need to scan all particle pairs for possible interactions but only those which are stored in the list. One can use this list as long as one of the particles has moved a distance of $\frac{s}{2}$. This criterion can be found for the worst case scenario that any two particles move directly towards each other with equal speed and no other particles kick them out of their paths. To check this criterion one has to calculate the displacement for each particle since the last calculation of the list, and compare the maximum of these displacements with $\frac{s}{2}$. This is only a minor calculation which can be done during the propagation of coordinates in the integration step.
The big advantage of the Verlet list is that it significantly reduces the overhead for searching interactions for many time steps. How big this improvement actually is depends again very strongly on the system parameters like r_{ia}, ρ and the time step of integration. For dense systems one can even weaken the $\frac{s}{2}$-criterion exploiting the fact that the worst case scenario drawn above is highly unlikely. For most of our simulations (at a density of $\rho = 0.85$) we were able to allow displacements of approximately $\frac{2s}{3}$ and thereby increase the number of reuses of a Verlet list by about 40% without changing our results. Of course, such non-exact optimizations have to be checked with great care.

3.1 Implementation of Periodic Boundary Conditions

In order to simplify the distance calculations while using periodic boundary conditions we employ the usual *ghost particle* scheme [5]. That is, the particles near the boundaries of the simulation box are replicated in memory with their coordinates shifted by the size of the simulation box. We store the pointers of these replicated particles within the cell lists instead of the original ones. This method saves us the complicated distance calculations across boundaries of the simulation box which would involve some backfolding of distance by multiples of the box size (usually done by costly integer conversions). Furthermore the concept of *ghost particles* fits neatly into the domain decomposition scheme

described in the next section.

4 Pair Interactions: Spatial Decomposition

The method of choice to parallelize MD with short-range forces seems to be spatial decomposition of the simulation box. That is, one simply splits up the box into smaller sub-boxes and assigns one processor for each sub-box to deal with it. Each processor stores and calculates only the coordinates, velocities and forces of those particles that currently are within its sub-box. Of course, since the particles are free to move in space they will eventually cross boundaries of these sub-boxes and therefore one has to redistribute them among the processors. Also when one has to calculate the forces acting on each particle a processor needs to know the coordinates of particles sitting near the boundary of its sub-box within other processors' domains. This information has to be updated at every step of the integration.
To solve the problem of boundary forces we introduce local *ghost particles* and *frames* around each sub-box. These *ghost frames* are equivalent to a shell around a sub-box with thickness $r_{ia} + s$ (see Fig. 4). The particles inside the frames are stored in the same array that holds the local (inner) particles of the sub-box. Therefore the array holding the coordinates has to be dimensioned big enough for local *and* ghost particles. Further we have to take into account that the numbers of particles fluctuate with time. Figure 3 depicts the sequence of storage of local and ghost particles within an array. First there are the local ones then follow the particles of the boundaries in $+X$ and $-X$ direction, thereafter the $+Y$ and $-Y$ boundaries and lastly the $+Z$ and $-Z$ frames.

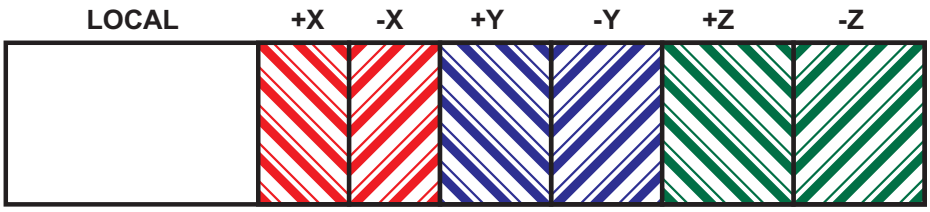

Figure 3: Storage order of different (local and frame) particles within the coordinate array.

This particular ordering follows directly from the sequence of communication steps for the boundaries. First each processor sorts out all local particles that belong to the X frames of their neighbors and copies them directly into the remote coordinate arrays. In the second step every processor searches for

particles in the Y boundaries including the particles in the X frames they have just received. In this way XY edges are sent automatically to their right destinations without the need of sending them explicitly in a separate communication step. In the last step we repeat this procedure for the Z boundaries and now the corners of the ghost frames reach their destination, too.

4.1 The Connectivity of Chains

So far we have not paid any attention to the fact that particles may not be identical. Since we want to simulate polymer chains and networks we need a method to deal with the connectivity between monomers. On a single CPU computer one would simply deal with the excluded volume and spring interactions separately. For the latter one can usually exploit the fact that the coordinates of the monomers of a single chain are stored linearly in memory. So one can handle the spring interactions within a single loop over all monomers of a chain. Using domain decomposition that ordering of particles in memory will be lost after the first few integration steps since particles move between the processors' domains. Therefore we give each particle a unique label which it carries along with itself during communication steps. In this label we have to store the following pieces of information: the identification number of the chain and the monomer index of the particle within that chain, and for networks also the index of the crosslink to which it is attached, or if it is itself a crosslink a flag which indicates this. All these numbers and flags fit into a single 64 bit integer again, which makes the computational task of finding the relationship between two monomers fairly easy and keeps the additional communication overhead very low.
For the implementation of the communication we employed the CRAY specific SHMEM-library. The main reasons for this choice were its easy applicability and its speed. The loss of portability is in our opinion not very severe, since a porting to MPI or PVM is straightforward and quick to implement. The details of the implementation are quite intricate at certain points. A detailed explanation will be given elsewhere [7].

5 Many-Body Interactions: A Hybrid Approach

The simple domain decomposition scheme in conjunction with the monomer labelling discussed above works very well for MD problems with pair interactions only. If one wants to simulate polymers on smaller length scales than the persistence length one needs to take into account the bending stiffness of the chains and at even smaller scales also rotational energy costs. These are usually incorporated into the models via the following types of potentials:

$$U_{bend} \quad = \quad C\left(\cos\Theta_i - \cos\Theta_0\right)^2, \tag{8}$$

$$U_{rot} = \sum_k c_k \cos^k \Psi_i, \tag{9}$$

$$\cos\Theta_i = \frac{\mathbf{u}_i \cdot \mathbf{u}_{i-1}}{u_i u_{i-1}}, \tag{10}$$

$$\cos\Psi_i = \frac{(\mathbf{u}_i \times \mathbf{u}_{i-1}) \cdot (\mathbf{u}_{i+1} \times \mathbf{u}_i)}{|\mathbf{u}_i \times \mathbf{u}_{i-1}|\,|\mathbf{u}_{i+1} \times \mathbf{u}_i|} \tag{11}$$

where $\mathbf{u}_i$ is the bond vector from monomer i to monomer $i+1$ along a chain and C, c_k and Θ_0 are model parameters. The bending term (8) is a three-body term, while the rotation potential (9) involves four particles. If we want to stick with domain decomposition, we would have to scan for triples and quadruples, which dramatically complicates the search loop. Again in single CPU simulations this problem would not be felt as a problem at all. Since all of the 3- and 4-body interactions are among monomers of a single chain one could handle them in one loop over all monomers of a chain exploiting the fact that monomer positions in memory are known at compile time.

To combine spatial decomposition with a linear storage of monomers of a single chain in memory we decided to use a double book-keeping scheme. On the one hand we deal with the excluded volume interactions as before by distributing single monomers onto the processors according to their coordinates. On the other hand we introduce a second set of particle coordinates which is however split into subsets on the basis of the chain index of the particles such that all monomers of a chain are treated by the same processor. The monomers of a single chain are stored in ascending order so that the calculation of spring, bending and rotational forces is easy to implement.

Of course, the double book-keeping has some disadvantages. The most important one is that we have to keep the two sets of coordinates coherent and also the forces calculated from the different sets have to be added in an additional communication step. It should be noted that, if no further constraints on the distribution of chains are imposed, these communication steps are *global* in the sense that probably each processor has to send data to every other processor in the set. Another (however much less severe) drawback is that the memory cost for each particle increases by approximately 30% (it is not doubled since not all arrays have to be replicated).

The following recipe summarizes the hybrid approach again and shows explicitly when and where the additional communication is needed:

```
CalcForces () {
  CalcExcludedVolumeForces();     /* using local + ghost particles */
  DistributeWholeChains();        /* each CPU collects the coords.
                                     for its chains */
  CalculateIntraChainForces();    /* using the linear chain arrays */
  AddIntraForcesToExlcudedVol(); /* Sort forces back from chain
                                     to domain order */
}
```

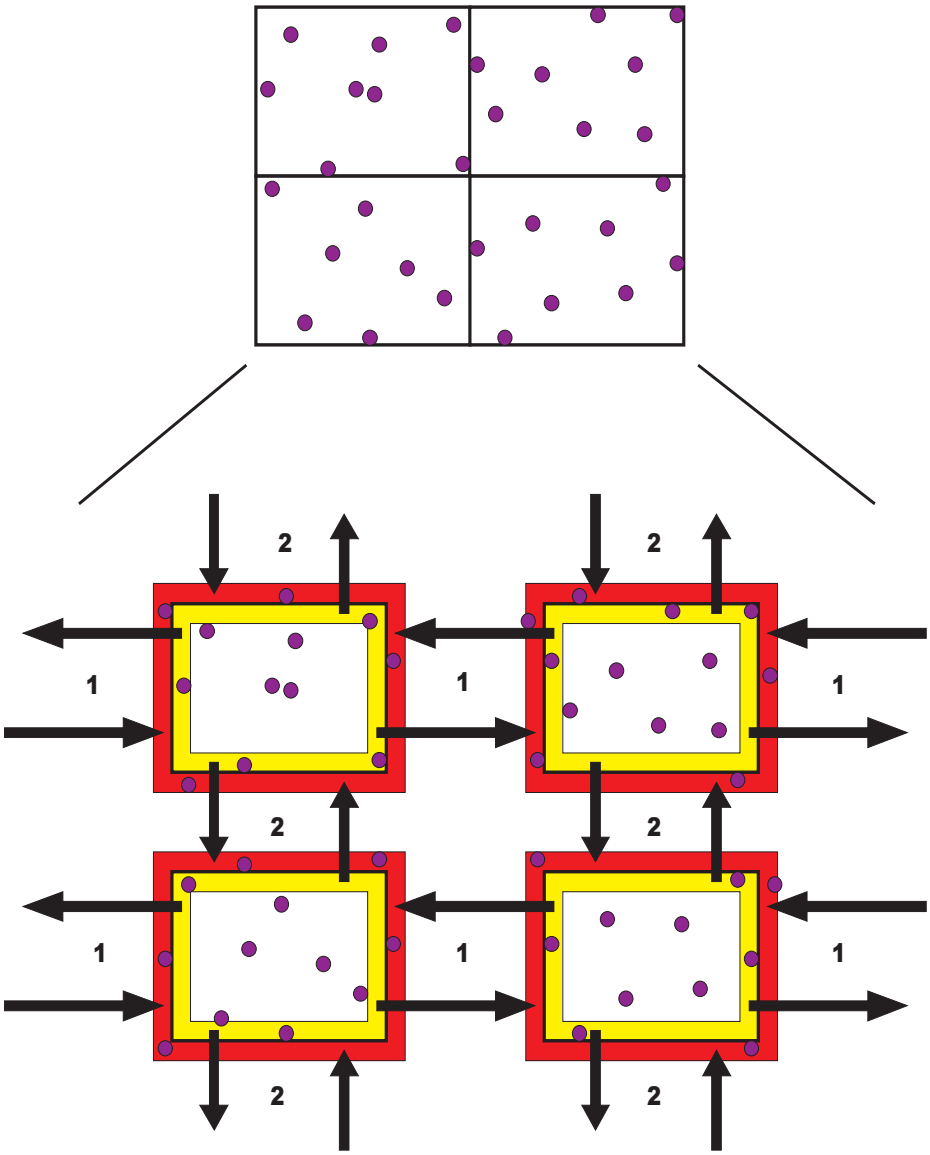

Figure 4: Spatial decomposition of the simulation box. *Ghost frames* (dark) are shown in the lower part; they are duplicates of the boundaries (light) on neighboring processors. The numbers indicate the sequence in which communication is performed.

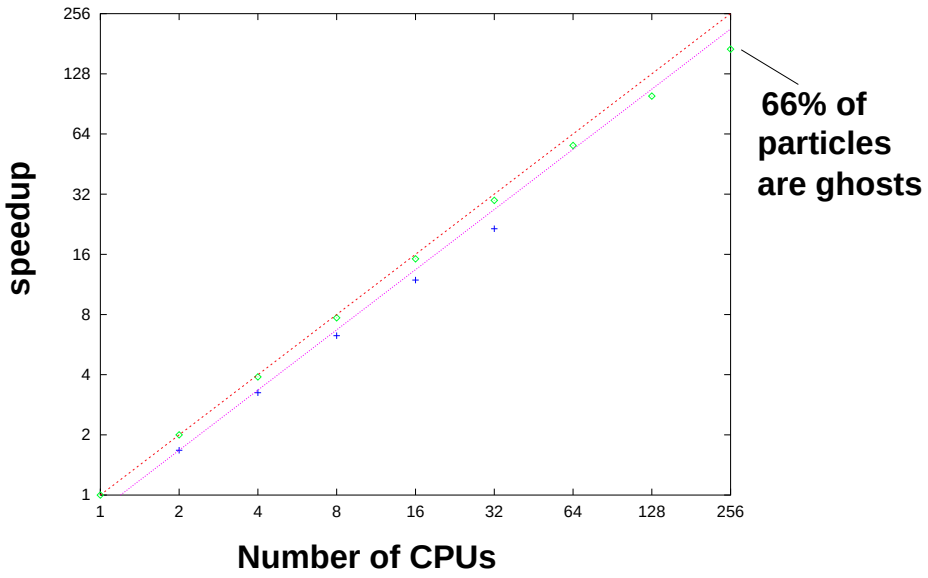

Figure 5: Speedup measurement of the two different algorithms. The diamonds are for the simple domain decomposition and the crosses show the hybrid algorithm results. The straight lines show the ideal scaling behavior to guide the eye.

6 Performance Analysis and Conclusion

We have tested the implementation of our MD program including the optimizations described in the previous sections for a typical system size on a CRAY T3E. For benchmarking we have chosen a system of 1000 flexible (no bending or torsional forces) chains of length 100 at a density of $\rho = 0.85$ at $T = 1.0$, $\Gamma = 0.5$, a time step of $h = 0.01$, a skin parameter $s = 0.38$ and a list criterion of $\max_i(r_{disp}(i,t,t_0)) \leq \frac{s}{1.4}$. The single CPU performance under these conditions was 148000 particle updates per second on one T3E node which can be compared to the to our knowledge fastest algorithm to date by Everaers et al. [6] running at 330000 particle updates per second on a single CRAY-YMP processor. One has to keep in mind that their grid search algorithm with neighbor cell assignment is fully vectorized and a T3E node is just a scalar 300 MHz DEC Alpha 21164 CPU.
Looking at the scaling of the code with the number of processors we observe very good behavior up to 256 processors for this benchmark (see Fig. 5), although more than $\frac{2}{3}$ of the particles on each CPU are ghost particles at this processor number. The speedup is still 175, i. e. approximately 70% of the ideal value.

The losses are almost completely due to load imbalance since there are too few particles (less than 400) in each processor's domain, giving rise to noticable fluctuations in the force calculations. The influence of increasing communication overhead is only a minor effect. For the investigation of inhomogeneities on polymer networks and gels we need systems in excess of 500000 particles, for which scaling is almost perfect up to 256 processors.
We have also tested our hybrid algorithm on the same benchmark to see the overhead compared to simple domain decomposition and its scaling behavior. The result is plotted in the same Fig. 5. In general we can see that the hybrid approach is about 15% to 20% slower and scaling starts to break down at about 64 processors. If one remembers that the hybrid code has been developed for the study of much more complex systems, the performance loss can be considered negligible. The relatively early breakdown in scaling is related to the fact that the communication process for switching between the domain decomposition picture and the chain decomposition picture involves communications between all processors and not just neighbouring ones. We believe that this can be overcome be refining the chain decomposition scheme in such a way that most of the chains treated by a processor are at least to their major parts within that processor's domain. This would render the communication local again.

Acknowledgements

Most of the development of the code has been done on the T3D and T3E systems at the RZG of the MPG in Garching, the T3E of the MPI for Polymer Research in Mainz and at the HLRZ in Jülich. We thank both the RZG and the HLRZ for a generous allocation of CPU time.

References

1. M. Doi, S. F. Edwards, *The Theory of Polymer Dynamics* (Clarendon Press, Oxford, 1986).
2. S. Panyukov, Y. Rabin, Physics Reports 269, 1 (1996).
3. *Monte Carlo and Molecular Dynamics Simulations in Polymer Science*, edited by K. Binder (Oxford University Press, New York, 1995).
4. M. P. Allen, D. J. Tildesley, *Computer Simulation of Liquids* (Clarendon, Oxford, 1987).
5. D. C. Rapaport, *The Art of Molecular Dynamics Simulation* (Cambridge University Press, New York, 1995).
6. R. Everaers, K. Kremer, Comp. Phys. Comm. 81, 19 (1994).
7. M. Pütz, A. Kolb, in preparation.