

Use of Neighbour Lists in Molecular Dynamics

S.M. Thompson

Introduction

The neighbour list is a device used in Molecular Dynamics simulations to increase execution speed by reducing the time taken to perform the most time consuming part of the calculation, the evaluation of the pair interactions. This article addresses the effect of some of the list parameters on the efficiency of utilisation of the list method. I will only consider systems of spherical molecules; the comments are general however and apply to other systems also.

Please note that all programming examples given here do not necessarily reflect the most efficient way of coding the particular sequence. The examples are for a system of Lennard-Jones atoms. The centre of mass coordinates are in the $(-1, +1)$ system, which is probably the most efficient coordinate system for use in molecular dynamics simulations.

Some of the comments in the text apply only to serial processors. In addition, although the method is discussed in the context of Molecular Dynamics, it is equally applicable to Monte Carlo. In fact it is probably even more effective in Monte Carlo simulations since it removes some of the inefficiencies inherent in the MC scheme. That, however, is another story.

The Neighbour List Method

For a system of N particles interacting with a pairwise additive potential, the force acting on each particle at each step of the simulation is usually evaluated by calling a subroutine (call it FORCE; see Figure 1) that examines all pairs of particles and evaluates the force between those pairs separated by less than a specified cutoff distance, RCUT. A loop over particles, $I=1, N-1$, encloses a second loop over the possible neighbours of that particle, $J=I+1, N$. For every one of the $N(N-1)/2$ pairs, the interparticle separation is evaluated, taking into account the usual boundary conditions. If this separation is greater than RCUT (which must be less than half the width of the box), the interaction force is assumed to be zero. Hence the J-loop is terminated, J is incremented and the process continues. If the pair separation is less than or equal to RCUT, the force is calculated according to some specified pair potential.

This procedure can be divided into two parts. The first part, the identification of those pairs separated by less than the cutoff distance, requires examination of all pairs, and so the computer time taken is proportional to the square of the number of particles. The second part, the evaluation of the forces for those pairs separated by less than the cutoff distance, requires computer time roughly proportional to the number of particles and the cube of the cutoff distance. Thus, for a given cutoff distance, the time taken to perform the first part of the calculation outweighs that taken to perform the second part, when the number of particles is reasonably large.

The neighbour list method implements a scheme whereby the first part of the calculation is reduced one order so that the time taken is proportional to the number of particles. The force

evaluation routine FORCE becomes two routines, FORCE1 and FORCE2 (see figures 2 and 3 respectively). FORCE1 proceeds in exactly the same way as FORCE, except that once the pair separations have been evaluated, they are compared with a chosen list parameter which we shall call the list radius, RLIST. The RLIST parameter is greater than the cutoff RCUT. If the separation is greater than RLIST, we can jump out of the J-loop as before. If it is less, then the subscript J is stored in a table called the neighbour list, and J is called a neighbour of I. The usual cutoff check then follows. Prior to entering the J-loop, the Ith entry of another table (call it NABIDX) is set with one more than the number of entries already made in the list at that point (the variable NELIST, which is initially zero). Each time an entry is made in the neighbour list, the number of entries is incremented. This scheme takes very little longer than the original force calculation in routine FORCE. If the maximum distance moved by any particle in the next, say, 10 steps is less than (RLIST - RCUT), then the neighbour list thus constructed contains all the required information about non-zero pair interactions for the next ten steps.

Subroutine FORCE2 calculates the pair forces using this list as follows. The usual I-loop is entered. The neighbours of I now have their subscripts (J) stored in locations NABIDX(I) through NABIDX(I+1)-1. The difference between these two numbers is usually much less than the number of particles in the system, and so the J-loop is less time consuming. The evaluation of forces now takes computer time proportional to the number of particles rather than to the square. If NABIDX(I) is greater than NABIDX(I+1)-1, then particle I has no neighbours and so the J-loop is not entered. This may be the case when simulating certain inhomogeneous or dilute systems.

When a new neighbour list is built using FORCE1, a vector for each particle is initialised to zero. At every subsequent step, this vector is incremented with the distance moved by each particle. The maximum distance moved, as determined at the end of every step, provides a guide as to when the list should be next updated. If the maximum distance moved is greater than FLUPD*(RLIST-RCUT), when FLUPD is a constant between zero and unity, then the list must be updated by calling FORCE1 again at the next step. If not, FORCE2 may be used again to calculate the forces.

Sometimes a constant list update interval, such as 10 steps, is used when implementing the neighbour list method. I do not recommend such a procedure, for two reasons. Firstly, if the simulation is performed at a state condition where particle motion is rapid, it is easy to get a condition where the list is used for a number of steps past the point where it should have been updated. This corrupts the particle trajectories and produces a discontinuous change in the energy of the system. Secondly, the list may well be used for less steps than it was possibly good for, with the result that a new neighbour list is built more times than is necessary. This slows down the program. It is better to let the program take full control over list management, in which case the list update interval is never of concern.

The parameter FLUPD deserves some thought. In cases where the list radius is only slightly greater than the cutoff, FLUPD will probably need to be in the region of 3/4. For larger list radii, a value up to about 0.95 can be used. In any event, FLUPD should be less than $(1-1/(NSLU+1))$, where NSLU is the shortest time (in steps) between successive updates of the list. Again, it is possible for the simulation to deal with this automatically, with the exception that an initial value

is required, and would be used for the first few list cycles. This specification for FLUPD relies on the temperature remaining more or less constant and there being no phase changes.

Thus we are left with the list radius RLIST as the only unspecified parameter influencing the performance improvements obtainable from the neighbour list.

Effect of the Neighbour List Radius

The value of RLIST will have little effect on the time taken to construct the neighbour list, since all pairs of particles have to be examined anyway. The larger the value of RLIST, the more table space will be required (approximately proportional to $RLIST^3$), and the less often it will need to be updated. Using a large value of RLIST, however, can lead to performance degradation, since the large size of the list means that the FORCE2 routine now has to do more work to sort out which entries have non-zero interactions. Consequently, there is an optimum list radius which gives maximum performance. This radius will depend to some extent on your definition of performance. If you have to pay for memory usage, then the optimum RLIST will be smaller than if you are aiming for maximum execution speed, due to the increased costs for memory usage offsetting the increase in execution speed.

Simulations of 256 and 500 Lennard-Jones atoms at a reduced density of 0.8 and at a reduced temperature of 0.76 were run for 1000 steps with the self-maintaining list mechanism as described above. The potential cutoff radius RCUT was 2.5 diameters. Different values of RLIST were used to obtain the data shown in the table. This data is plotted in Figures 4 and 5, where RLIST is in units of collision diameters.

It can be seen that the optimum neighbour list radius, in terms of execution speed, is approximately 0.2 diameters larger than the cutoff radius for $N = 256$, and approximately 0.4 diameters larger for $N = 500$. The CPU time required per step falls steeply as the list is "turned on", and increases again, but more slowly, as the list radius becomes larger than the optimum. However, for list radii around the optimum, there is not very much effect of varying RLIST on the execution speed, especially for the $N = 500$ system. It therefore appears from this data that 2.7 diameters would provide good performance for both systems. The optimum list radius increases execution speed by a factor of 1.54 for $N = 256$, and by over a factor of 2 for $N = 500$. For even larger systems, the effect on the execution time of introducing the list is much more dramatic. It remains to be seen what the effect of density is on these results.

List Radius	NSAVU (*)	N = 256	N = 500 Time (**)
- no list -	n/a	3 . 33	10 . 00
2 . 60	5 . 78	2 . 24	4 . 93
2 . 70	12 . 50	2 . 17	4 . 55
2 . 90	26 . 32	2 . 28	4 . 51
3 . 10	43 . 48	2 . 47	4 . 79
3 . 43	83 . 33	2 . 89	—
3 . 50	100 . 00	—	5 . 86

Notes:

- (*) NSAVU is the average number of steps between updates of the neighbour list. This data is for the $N = 256$ simulation (except for the last entry), but this figure is essentially independent of the system size. It does, however, depend on the value of the FLUPD parameters.
- (**) "Time" is the CPU time required per step. These times were obtained using a PDP 11/70 minicomputer running the RSX-11M V4.0 operating system. The program was compiled with the PDP-11 FORTRAN-77 compiler, version 4.1, using the /NOTRACE option. This machine is approximately 18% the speed of an IBM 370/168. CPU times were measured using an in-house performance measurement package.

```
DO 100 I = 1,255
  JBEG = I + 1
  DO 200 J = JBEG,256
    XDIFF = XO(I) - XO(J)
    YDIFF = YO(I) - YO(J)
    ZDIFF = ZO(I) - ZO(J)
    XDIFF = XDIFF - 2*INT(XDIFF)
    YDIFF = YDIFF - 2*INT(YDIFF)
    ZDIFF = ZDIFF - 2*INT(ZDIFF)
    RSQ = XDIFF*XDIFF + YDIFF*YDIFF + ZDIFF*ZDIFF
    IF(RSQ.GT.RCUTSQ)GO TO 200
    RSQI = 1.0/RSQ
    R6 = SIGMA6*RSQI*RSQI*RSQI
    DUDRR1 = 48.0*RSQI*R6*(R6 - 0.5)
    FXIJ = XDIFF*DUDRR1
    FYIJ = YDIFF*DUDRR1
    FZIJ = ZDIFF*DUDRR1
    FX(I) = FX(I) + FXIJ
    FY(I) = FY(I) + FYIJ
    FZ(I) = FZ(I) + FZIJ
    FX(J) = FX(J) - FXIJ
    FY(J) = FY(J) - FYIJ
    FZ(J) = FZ(J) - FZIJ
  200  CONTINUE
100  CONTINUE
```

Figure 1: Extract from routine FORCE

```
NELIST = 0
DO 100 I = 1,255
  NABIDX(I) = NELIST + 1
  JBEG = I + 1
  DO 200 J = JBEG,256
    XDIFF = XO(I) - XO(J)
    YDIFF = YO(I) - YO(J)
    ZDIFF = ZO(I) - ZO(J)
    XDIFF = XDIFF - 2*INT(XDIFF)
    YDIFF = YDIFF - 2*INT(YDIFF)
    ZDIFF = ZDIFF - 2*INT(ZDIFF)
    RSQ = XDIFF*XDIFF + YDIFF*YDIFF + ZDIFF*ZDIFF
    IF(RSQ.GT.RLSTSQ)GO TO 200
    NELIST = NELIST + 1
    NABTAB(NELIST) = J
    IF(RSQ.GT.RCUTSQ)GO TO 200
    RSQI = 1.0/RSQ
    R6 = SIGMA6*RSQI*RSQI*RSQI
    DUDRR1 = 48.0*RSQI*R6*(R6 - 0.5)
    FXIJ = XDIFF*DUDRR1
    FYIJ = YDIFF*DUDRR1
    FZIJ = ZDIFF*DUDRR1
    FX(I) = FX(I) + FXIJ
    FY(I) = FY(I) + FYIJ
    FZ(I) = FZ(I) + FZIJ
    FX(J) = FX(J) - FXIJ
    FY(J) = FY(J) - FYIJ
    FZ(J) = FZ(J) - FZIJ
  200 CONTINUE
100 CONTINUE
  NABIDX(256) = NELIST + 1
```

Figure 2: Extract from routine FORCE1

```
DO 100 I = 1,255
  JBEG = NABIDX(I)
  JEND = NABIDX(I+1) - 1
  IF(JBEG.GT.JEND)GO TO 100
  DO 200 JX = JBEG,JEND
    J = NABTAB(JX)
    XDIFF = XO(I) - XO(J)
    YDIFF = YO(I) - YO(J)
    ZDIFF = ZO(I) - ZO(J)
    XDIFF = XDIFF - 2*INT(XDIFF)
    YDIFF = YDIFF - 2*INT(YDIFF)
    ZDIFF = ZDIFF - 2*INT(ZDIFF)
    RSQ = XDIFF*XDIFF + YDIFF*YDIFF + ZDIFF*ZDIFF
    IF(RSQ.GT.RCUTSQ)GO TO 200
    RSQI = 1.0/RSQ
    R6 = SIGMA6*RSQI*RSQI*RSQI
    DUDRR1 = 48.0*RSQI*R6*(R6 - 0.5)
    FXIJ = XDIFF*DUDRR1
    FYIJ = YDIFF*DUDRR1
    FZIJ = ZDIFF*DUDRR1
    FX(I) = FX(I) + FXIJ
    FY(I) = FY(I) + FYIJ
    FZ(I) = FZ(I) + FZIJ
    FX(J) = FX(J) - FXIJ
    FY(J) = FY(J) - FYIJ
    FZ(J) = FZ(J) - FZIJ
  200 CONTINUE
100 CONTINUE
```

Figure 3: Extract from routine FORCE2

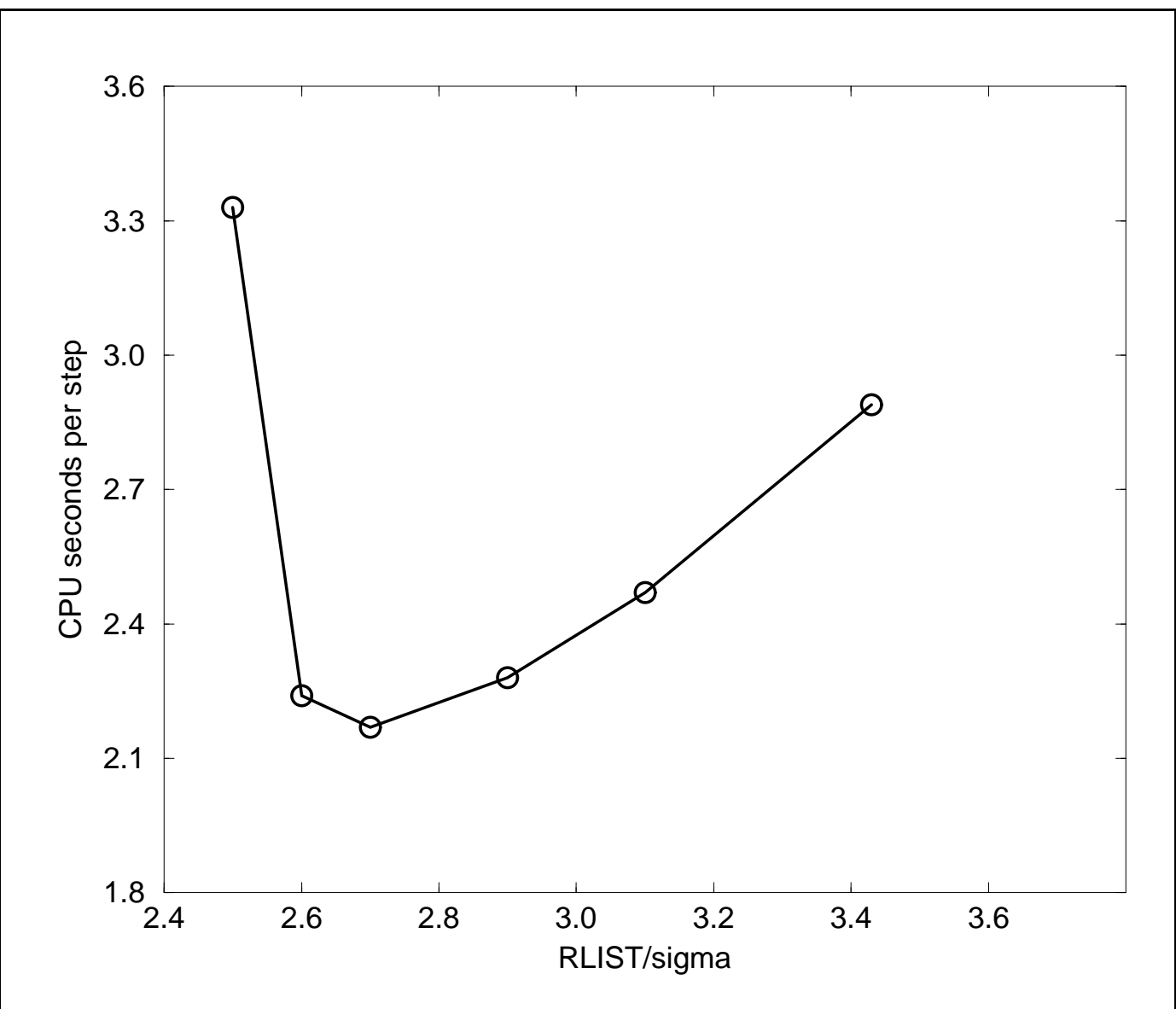


Figure 4: CPU time vs. list radius, $N = 256$

