law. However, there is a further problem. The force between a pair of molecules is still discontinuous at $r_{ij} = r_c$. For example, in the Lennard-Jones case, the force is given by eqn (5.3) for $r_{ij} \leqslant r_c$, but is zero for $r_{ij} > r_c$. The magnitude of the discontinuity is $0.039\varepsilon\sigma^{-1}$ for $r_c = 2.5\sigma$. It can cause instability in the numerical solution of the differential equations. To avoid this difficulty, a number of workers have used a 'shifted-force potential' [Stoddard and Ford 1973; Streett, Tildesley, and Saville 1978a; Nicolas *et al.* 1979; Powles *et al.* 1982]. A small linear term is added to the potential, so that its derivative is zero at the cutoff distance

$$v^{SF}(r_{ij}) = \begin{cases} v(r_{ij}) - v_c - \left(\dfrac{dv(r_{ij})}{dr_{ij}}\right)_{r_{ij}=r_c} (r_{ij} - r_c) & r_{ij} \leqslant r_c \\ 0 & r_{ij} > r_c \end{cases} \tag{5.13}$$

The discontinuity now appears in the gradient of the force, not in the force itself. The shifted-force potential for the Lennard-Jones case is shown in Fig. 5.3. The force goes smoothly to zero at the cutoff r_c, removing problems in energy conservation and any numerical instability in the equations of motion. Making the additional term quadratic [Stoddard and Ford 1973] avoids taking a square root. Of course, the difference between the shifted-force potential and the original potential means that the simulation no longer corresponds to the desired model liquid. However, the thermodynamic properties of a fluid of particles interacting with the unshifted potential can be recovered from the shifted-force potential simulation results, using a simple perturbation scheme [Nicolas *et al.* 1979; Powles 1984b].

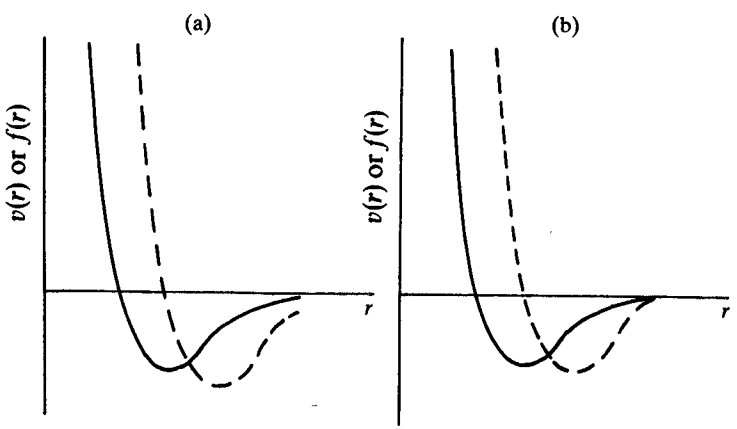


Fig. 5.3 Magnitude of the pair potential (solid line) and force (dashed line) for (a) the Lennard-Jones potential and (b) its shifted-force modification.

5.3 Neighbour lists

In the inner loops of the MD and MC programs, we consider a molecule i and loop over all molecules j to calculate the minimum image separations. If

molecules are separated by distances greater than the potential cutoff, the program skips to the end of the inner loop, avoiding expensive calculations, and considers the next neighbour. In this method, the time to examine all pair separations is proportional to N^2. Verlet [1967] suggested a technique for improving the speed of a program by maintaining a list of the neighbours of a particular molecule, which is updated at intervals. Between updates of the neighbour list, the program does not check through all the j molecules, but just those appearing on the list. The number of pair separations explicitly considered is reduced. This saves time in looping through j, minimum imaging, calculating r_{ij}^2, and checking against the cutoff, for all those particles not on the list. Obviously, there is no change in the time actually spent calculating the energy and forces arising from neighbours within the potential cutoff. In this section, we describe some useful time-saving neighbour list methods. These methods are equally applicable to MC and MD simulations, and for convenience we use the MD method to illustrate them. There are differences concerning the relative sizes of the neighbour lists required in MC and MD and we return to this point at the end of the next section. Related techniques may be used to speed up MD of hard systems [Erpenbeck and Wood 1977]. In this case, the aim is to construct and maintain, as efficiently as possible, a table of future collisions between pairs of molecules. The entire question of the scheduling of molecular collisions has been discussed by Rapaport [1980].

5.3.1 The Verlet neighbour list

In the original Verlet method, the potential cutoff sphere, of radius r_c, around a particular molecule is surrounded by a 'skin', to give a larger sphere of radius r_l, as shown in Fig. 5.4. At the first step in a simulation, a list is constructed of all the neighbours of each molecule, for which the pair separation is within r_l. These neighbours are stored in a large array, called, shall we say, LIST. LIST is quite large, of dimension roughly $4\pi r_l^3 \rho N/6$. At the same time a second indexing array, POINT, of size N, is constructed. POINT(I) points to the position in the array LIST where the first neighbour of molecule I can be found. Since POINT(I + 1) points to the first neighbour of molecule I + 1, then POINT(I + 1) − 1 points to the last neighbour of molecule I. Thus, using POINT, we can readily identify the part of the large LIST array which contains neighbours of I. The code for setting up the arrays LIST and POINT is given in program F.19.

Over the next few time steps, the list is used in the force/energy evaluation routine. For each molecule I, the program identifies the neighbours J, by running over LIST from POINT(I) to POINT(I + 1) − 1. It is essential to check that POINT(I + 1) is actually greater than POINT(I): if this is not the case, then molecule I has no neighbours, and can be skipped. This is certainly possible in dilute systems. A sample force routine using the Verlet list is given in program F.19. From time to time, the neighbour list is reconstructed, and

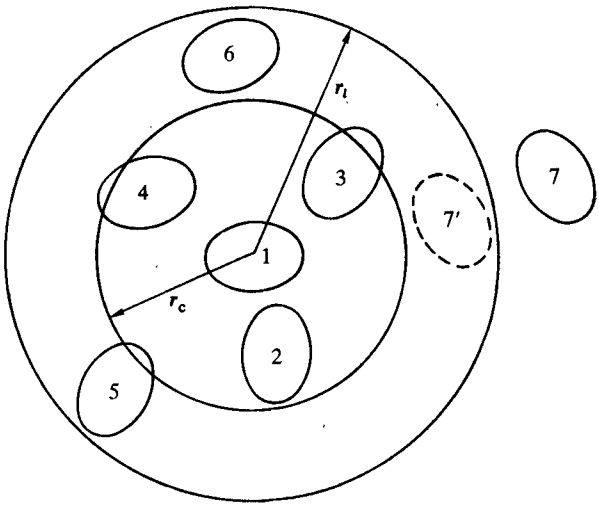


Fig. 5.4 The cutoff sphere, and its skin, around a molecule 1. Molecules 2, 3, 4, 5, and 6 are on the list of molecule 1; molecule 7 is not. Only molecules 2, 3, and 4 are within the range of the potential at the time the list is constructed.

the cycle is repeated. The algorithm is successful because the skin around r_c is chosen to be thick enough so that between reconstructions a molecule, such as 7 in Fig. 5.4, which is not on the list of molecule 1, cannot penetrate through the skin into the important r_c sphere. Molecules such as 3 and 4 can move in and out of this sphere, but since they are on the list of molecule 1, they are always considered regardless, until the list is next updated.

The interval between updates of the table is often fixed at the beginning of the program, and intervals of 10–20 steps are quite common. An important refinement allows the program to update the neighbour list automatically. When the list is constructed, a vector for each molecule is set to zero. At subsequent steps, the vector is incremented with the displacement of each molecule. Thus it stores the total displacement for each molecule since the last update. When the sum of the magnitudes of the two largest displacements exceeds $r_l - r_c$, the neighbour list should be updated again [Fincham and Ralston 1981; Thompson 1983]. The code for automatic updating of the neighbour list is given on microfiche F.19.

The list sphere radius, r_l, is a parameter that we are free to choose. As r_l is increased, the frequency of updates of the neighbour list will decrease. However, with a large list, the efficiency of the non-update steps will decrease. This balance has been examined by Thompson [1983] for MD simulations of 256 and 500 Lennard-Jones atoms. Simulations at $\rho^* = 0.8$, $T^* = 0.76$, were run for 1000 time steps; r_c was fixed at 2.5σ, and r_l varied. The results are given in Table 5.1. A list cutoff of about 2.7σ would provide substantial speed

Table 5.1 *Time saving using a Verlet neighbour list* [*Thompson* 1983].

List Radius	Update[a] interval	Time[b] $N = 256$	Time[b] $N = 500$
no list	—	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

[a]Update interval is the average number of steps between updates. It is essentially independent of system size.
[b]Time is CPU time per step, in seconds. The runs were performed on a PDP 11/70.

increases for both systems, but for systems of size $N \approx 500$ and larger, the improvement is dramatic. As the size of the system becomes larger, the size of the LIST array grows, approximately $\propto N$. If storage is a priority, then a binary representation of the list can be employed [O'Shea 1983]. Each bit in a two-dimensional array represents a pair of molecules i and j. This bit is set to 1 if the molecules are neighbours, and zero otherwise. The array is used to check for neighbours at subsequent steps, and is revised at suitable intervals. For a system of 256 molecules, a conventional neighbour list would require approximately 64×256 words; on a 16-bit machine, the binary array reduces this to 8×256 words.

In the MC method, the array POINT has a size $N + 1$ rather than N, since the index I runs over all N atoms rather than $N - 1$ as in MD. In addition, the array LIST is roughly twice as large in MC as in a corresponding MD program. In the MD technique, the list for a particular molecule i contains only the molecules j with an index greater than i, since in this method we use Newton's third law to calculate the force on j from i at the same time as the force on i from j. In the MC method particles i and j are moved independently and the list must contain separately the information that i is a neighbour of j and j a neighbour of i. In this case the binary representation discussed by O'Shea [1983] is particularly useful.

5.3.2 Cell structures and linked lists

As the size of the system increases towards 1000 molecules, the conventional neighbour list becomes too large to store easily, and the logical testing of every

pair in the system is inefficient. An alternative method of keeping track of neighbours for large systems is the cell index method [Quentrec and Brot 1975; Hockney and Eastwood 1981]. The cubic simulation box (extension to non-cubic cases is possible) is divided into a regular lattice of M × M × M cells. A two-dimensional representation of this is shown in Fig. 5.5. These cells are chosen so that the side of the cell $l = L/M$ is greater than the cutoff distance for the forces, r_c. For the two-dimensional example of Fig. 5.5, the neighbours of any molecule in cell 13 are to be found in the cells 7, 8, 9, 12, 13, 14, 17, 18, 19. If there is a separate list of molecules in each of those cells, then searching through the neighbours is a rapid process. For the two-dimensional system illustrated, there are approximately $N_c = N/M^2$ molecules in each cell; the analogous result in three dimensions would be $N_c = N/M^3$. Using the cell

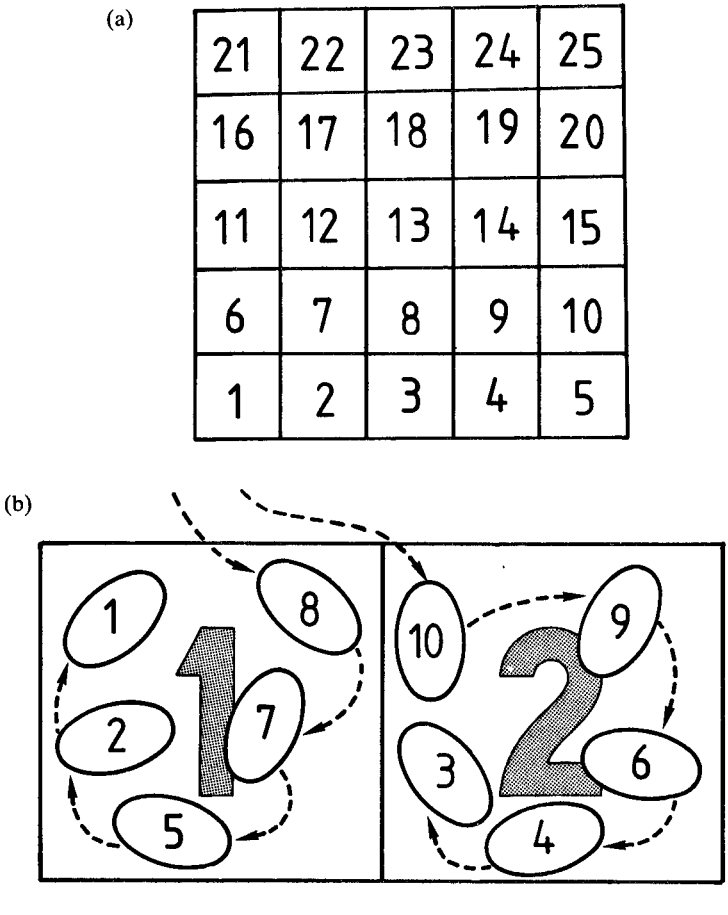
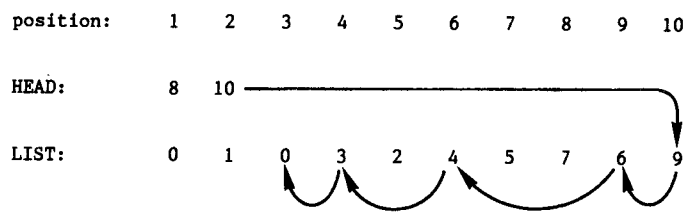


Fig. 5.5 The cell method in two dimensions. (a) The central box is divided into $M \times M$ cells ($M = 5$). (b) A close-up of cells 1 and 2, showing the molecules and the link-list structure.

structure in two dimensions we need only examine $9NN_c$ pairs (or just $4.5NN_c$ if we take advantage of the third law in the MD method). This contrasts with N^2 (or $\frac{1}{2}N(N-1)$) for the brute force approach. When the cell structure is used in three dimensions, then we compute $27NN_c$ interactions ($13.5NN_c$ for MD) as compared with N^2 (or $\frac{1}{2}N(N-1)$).

The cell structure may be set up and used by the method of linked lists [Knuth 1973, Chapter 2; Hockney and Eastwood 1981, Chapter 8]. The first part of the method involves sorting all the molecules into their appropriate cells. This sorting is rapid, and may be performed every step. Two arrays are created during the sorting process. The 'head-of-chain' array (HEAD) has one element for each cell. This element contains the identification number of one of the molecules sorted into that cell. This number is used to address the element of a linked-list array (LIST), which contains the number of the next molecule in that cell. In turn, the LIST array element for that molecule is the index of the next molecule in the cell, and so on. If we follow the trail of link-list references, we will eventually reach an element of LIST which is zero. This indicates that there are no more molecules in that cell, and we move on to the head-of-chain molecule for the next cell. To illustrate this, imagine a simulation of particles in two cells: 1, 2, 5, 7, and 8 in cell one and 3, 4, 6, 9, and 10 in cell two (see Fig. 5.5). The HEAD and LIST arrays are



For cell two, HEAD(2) = 10, and the route through the linked list is arrowed. The construction of HEAD and LIST is straightforward. We take the usual unit cube simulation box, in which case the cell length is just $1/M$. The inverse of this quantity is stored in CELLI.

```
      DO 100 ICELL = 1, NCELL

         HEAD(ICELL) = 0

100   CONTINUE

      CELLI = REAL ( M )

      DO 200 I = 1, N

         ICELL = 1 + INT ( ( RX(I) + 0.5 ) * CELLI )
     :             + INT ( ( RY(I) + 0.5 ) * CELLI ) * M
     :             + INT ( ( RZ(I) + 0.5 ) * CELLI ) * M * M
         LIST(I) = HEAD(ICELL)
         HEAD(ICELL) = I

200   CONTINUE
```

In MD, the calculation of the forces is performed by looping over all cells. For a given cell, the program sorts through the linked list. A particular molecule on the list may interact with all molecules in the same cell that are further down the list. This avoids counting the *ij* interaction twice. A particular molecule also may interact with the molecules in the neighbouring cells. To avoid double counting of these forces, only a limited number of the neighbouring cells are considered. This idea is most easily explained with reference to our two-dimensional example shown in Fig. 5.5. A molecule in cell 13 interacts with eight neighbouring cells, but the program only checks the chains in cells 9, 14, 18, and 19. Interactions between cells 13 and 12 are checked when cell 12 is the focus of attention, and so on. In this way, we can make full use of Newton's third law in calculating the forces. We note that, for a cell at the edge of the basic simulation box, (15 for example in Fig. 5.5) it is necessary to consider the periodic cells (6, 11, and 16). An example of the code for constructing and searching through cells is given in F.20. The cell structure may be used somewhat more efficiently if the molecules in each cell are sorted into order of increasing (say) x-coordinate [Hockney and Eastwood 1981].

The linked-list method can also be used with the MC technique. In this case, a molecule move involves checking molecules in the same cell, and in all the neighbouring cells. The linked-list method has been used with considerable success in simulation of systems such as plasmas, galaxies, and ionic crystals, which require a large number of particles [Hockney and Eastwood 1981]. In both the linked-list methods and the Verlet neighbour list, the computing time required to run a simulation tends to increase linearly with the number of particles N rather than quadratically. This rule of thumb does not take into account the extra time required to set up and manipulate the lists. For small systems ($N \approx 100$) the overheads involved make the use of lists unprofitable.

A modification of the cell structure employs cells that are sufficiently small that at most one particle can occupy each cell. In this case, a linked-list structure as described above is not required: the program simply loops over all cells, and conducts an inner loop over the cells that are within a distance r_c of the cell of interest. Advantages of this method are that the list of cells 'within range' of each given cell may be computed at the start of the program, remaining unaltered throughout, and that a simple decision (is the cell occupied or not?) is required at each stage.

5.4 Multiple time step methods

In a typical MD simulation, as much as 95 per cent of the computing time is spent in examining the complete set of $\frac{1}{2}N(N-1)$ pairs, identifying those pairs separated by less than the cutoff distance r_c, and computing the forces for this subset. The remaining pair interactions do not influence the dynamics of the system. For each molecule, the set of neighbours within the cutoff changes with

time; the task of identifying and rejecting the others is time-consuming, but the book-keeping methods discussed in the previous sections reduce this time considerably. Any further increase in speed can only be achieved by cutting down the time spent actually evaluating forces between pairs within cutoff range.

The multiple time-step method [Streett, Tildesley, and Saville 1978a, b] is designed to do this. Figure 5.6 shows the close neighbours of an atom i, which are split into two groups: 'primary' neighbours, which lie within a distance r_p of atom i, and 'secondary' neighbours, which are at a distance between r_p and r_c from i. In this way, the total force $\mathbf{f}_i$ on i is separated into a primary component $\mathbf{f}_i^p$ due to the close neighbours, and a secondary component $\mathbf{f}_i^s$ from the more remote neighbours. Typically, for a Lennard-Jones fluid, r_p would be chosen to lie between σ and 1.5σ. The motion of atom i is dominated by a rapidly changing primary force resulting from collisions with the nearest neighbour 'cage'. The secondary force is smaller, and changes more slowly with time. The multiple time step method takes advantage of this separation, and aims to calculate the secondary pair interactions less frequently than the more important primary ones. The method is most conveniently used in conjunction with the Gear predictor–corrector algorithm (Section 3.2 and Appendix E).

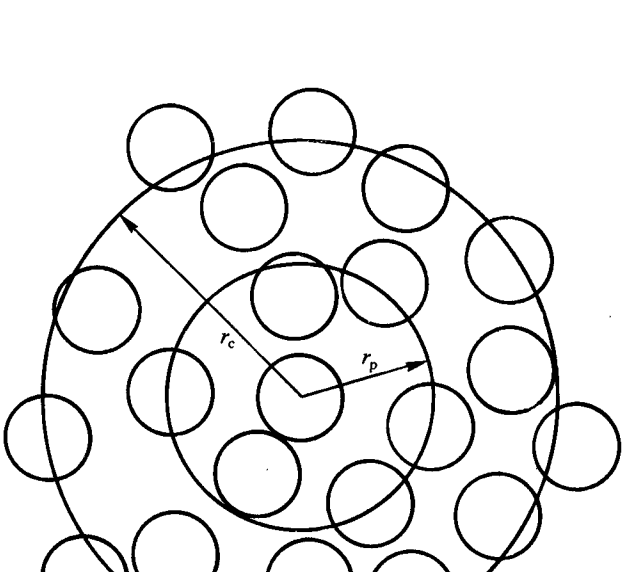


Fig. 5.6 The close neighbours of a molecule. These molecules are divided into primary and secondary neighbours.

At time t the primary and secondary forces on each atom i are calculated in the usual way. At the same time, the time derivatives of the secondary forces, $\dot{\mathbf{f}}_i^s(t)$, $\ddot{\mathbf{f}}_i^s(t) \ldots$, up to order m say, are calculated, and a list of primary neighbours is compiled. At each of the next $\tau_{mts} - 1$ steps, the primary force is computed explicitly using the list. The secondary force is estimated using a Taylor series of order m:

$$\mathbf{f}_i^s(t + \tau\delta t) \approx \mathbf{f}_i^s(t) + (\tau\delta t)\dot{\mathbf{f}}_i^s(t) + \tfrac{1}{2}(\tau\delta t)^2 \ddot{\mathbf{f}}_i^s + \ldots . \tag{5.14}$$

Following these τ_{mts} steps (one involving a full calculation of forces and the secondary time derivatives, and $\tau_{mts} - 1$ using eqn (5.14) for the secondary forces) the entire process is repeated, starting with a recalculation of the primary list. Thus, the method effectively uses two time steps: δt for the primary interactions and $\tau_{mts}\delta t$ for the secondary ones.

Convenient expressions for the force derivatives can be readily obtained in terms of the time derivatives of particle positions and spatial derivatives of the potential. For example,

$$\dot{\mathbf{f}}_i^s = \sum_j A_{ij}\dot{\mathbf{r}}_{ij} + B_{ij}(\mathbf{r}_{ij}\cdot\dot{\mathbf{r}}_{ij})\mathbf{r}_{ij} \tag{5.15}$$

and

$$\ddot{\mathbf{f}}_i^s = \sum_j [B_{ij}(\mathbf{r}_{ij}\cdot\ddot{\mathbf{r}}_{ij} + \dot{\mathbf{r}}_{ij}\cdot\dot{\mathbf{r}}_{ij}) + C_{ij}(\mathbf{r}_{ij}\cdot\dot{\mathbf{r}}_{ij})^2]\mathbf{r}_{ij} + 2B_{ij}(\mathbf{r}_{ij}\cdot\dot{\mathbf{r}}_{ij})\dot{\mathbf{r}}_{ij} + A_{ij}\ddot{\mathbf{r}}_{ij} \tag{5.16}$$

where $A_{ij} = -(1/r_{ij})(\mathrm{d}v_{ij}/\mathrm{d}r_{ij})$, $B_{ij} = (1/r_{ij})(\mathrm{d}A_{ij}/\mathrm{d}r_{ij})$, and $C_{ij} = (1/r_{ij})(\mathrm{d}B_{ij}/\mathrm{d}r_{ij})$, and the summations range over all secondary neighbours j. These calculations take place in the force loop; time derivatives such as $\ddot{\mathbf{r}}_{ij} = \ddot{\mathbf{r}}_i - \ddot{\mathbf{r}}_j$ are readily calculated from the time derivatives of atomic positions available from the predictor stage of the Gear algorithm (Section 3.2). An example of a program using this method is given in F.21.

In the study of Streett *et al.* [1978a] a third-order Taylor series, with $\tau_{mts} = 10$, $r_c = 2.5\sigma$ and $r_p = 1.1\sigma$ was used. In the simulation of a Lennard-Jones fluid at a reduced density $\rho^* = 0.8$, the average numbers of primary and secondary neighbours were found to be 1.4 and 24, respectively. The multiple time-step method in this case resulted in a speed improvement of a factor of 7–10 over a conventional MD program with no neighbour lists, and a factor of 3–5 over a program using a Verlet neighbour list. These relative timings will depend upon precise details of operations carried out in the force loop, and may be machine dependent.

The method has been successfully applied to molecular fluids [Streett, *et al.* 1978b] and to chain molecules [Swindoll and Haile 1984]. It has been extended to the case when there are three regions inside the r_c sphere, each with a different time step [Nicolas *et al.* 1979]. One interesting application of the method is in the simulation of fluids with three-body forces [Haile 1978]. For

the Axilrod–Teller potential (see Appendix C and Maitland *et al.* [1981]), the three-body contribution to the force varies slowly with time, and it is possible to treat the entire three-body component as a secondary interaction. In this way, the prohibitively expensive summation over triplets may be avoided for most of the steps in the simulation.

There are still some difficulties with the method. Firstly, there is a considerable programming effort required to incorporate the relevant code into a conventional program. Secondly, the method has only been used with a time step δt rather smaller than that used in conventional simulations (e.g. 5×10^{-15} s [Streett *et al.* 1978a], as opposed to the more usual 10^{-14} [Verlet 1967]). It is found that the use of a larger time step requires an increase in r_p and a decrease in the update interval τ_{mts}, thus offsetting some of the advantages gained by using multiple time steps. The efficiency of the method is clearly very sensitive to the choice of r_p; perhaps the best way to test this method is to fix the desired time step, decide upon a satisfactory level of energy conservation, and adjust r_p and the update interval τ_{mts}, to maximize the efficiency of the simulation subject to these constraints. For molecular fluids, where a shorter time step is often required in any case, these difficulties are less evident.

5.5 How to handle long-range forces

5.5.1 Introduction

In the previous sections, we have discussed the core of the program when the forces are short ranged. In this section, we turn our attention to the handling of long-range forces in simulations.

A long-range force is often defined as one in which the spatial interaction falls off no faster than r^{-d} where d is the dimensionality of the system. In this category are the charge–charge interaction between ions ($v^{zz}(r) \sim r^{-1}$) and the dipole–dipole interaction between molecules ($v^{\mu\mu}(r) \sim r^{-3}$). These forces are a serious problem for the computer simulator, since their range is greater than half the box length for a typical simulation of ≈ 500 molecules. The brute force solution to this problem would be to increase the size of the central box L to hundreds of nanometres so that the screening by neighbours would diminish the effective range of the potentials. Even with modern computers, this solution is impracticable, since the time required to run such a simulation is approximately proportional to N^2, i.e. L^6.

How can such potentials be handled? The problem is particularly acute for $v^{zz}(r)$. Straightforward spherical truncation of the potential can be ruled out. The resulting sphere around a given ion could be charged, since the number of anions and cations need not balance at any instant. The tendency of ions to migrate back and forth across the spherical surface would create artificial effects at $r = r_c$. This can be countered by distributing a new charge over the