

Neighbour Lists Again

Seamus F. O'Shea
Department of Chemistry
University of Lethbridge
Lethbridge, Alberta
Canada T1K 3M4

In a recent article, Thompson [1] described the use of neighbour lists in simulations, giving a detailed account of the implementation and the resulting economies. This note deals with a simple variant on the method, and discusses the choice of neighbour list technique, if any, to be used in various circumstances.

The idea was originally introduced in the classic paper of Verlet [2] on the simulation of a dense Lennard-Jones fluid. Using a cut-off radius slightly greater than the truncation radius of the interaction potential [1], a list is made for each molecule of all other molecules which are, or could soon be, within interaction range. The lists are renewed at intervals [1], and during the intervening steps only the listed pairs are considered in calculating the pair interaction energies and forces. With a cut-off radius equal to half the (cubic) box length only about half of the distinct pairs need be listed, and the resulting savings in computer time grows rapidly with N because the number of pairs is $N(N-1)/2$. However, the conventional method requires that an integer index be stored for every active pair, so that the list grows roughly as $N^2/4$. This can quickly become burdensome in machines with limited storage, or where costs or priority depend on the amount of storage used. The alternative implementation described here combines the efficiency inherent in the use of neighbour lists with compact storage.

For a given molecule, any other molecule is either within the cut-off range, or it is not, and this situation is ideally suited to a binary representation. The first form described here is particularly suitable for conventional Monte Carlo simulations, where the use of single particle moves requires that one be able to identify all neighbours of the molecule being moved. For molecular dynamics simulations, where all active pair interactions are required at each step, a simple and obvious modification can be used to reduce the required storage by another factor of two, and the coding is even simpler. FORTRAN, still the dominant medium for simulation code, had no bit-handling facilities under the old standard, and rumour has it that the implementation of the new standard is not yet uniform. Many compilers offer non-standard extensions for manipulating bit strings, but the details vary from site to site. For these reasons the code given here is more cumbersome than it need be, but it should be easily adapted to run on any compiler, and those with enlightened compilers can use more direct implementations.

The neighbour lists are held in a two-dimensional array, LIST, of dimension $NWORD \times N$. The column length, NWORD, is the smallest integer number of words capable of holding N bits, given a wordlength of NBITS.

```
NWORD=N/NBITS
IF (MOD(N,NBITS).NE.0) NWORD=NWORD+1
```

LIST must be cleared before the lists are established or revised, and this is easily done by setting each word in LIST to .FALSE.

```
DO 1 J=1,N
  DO 1 I=1,NWORD
1  LIST(I,J)=.FALSE.
```

Before creating the lists for the first time, a LOGICAL array MASK

```
MASK(1)      1000 ... 0
MASK(2)      0100 ... 0
MASK(3)      0010 ... 0
MASK(4)      0001 ... 0
MASK(NBITS)  0000 ... 1
```

must be generated; the easiest way to do it is to treat MASK as an integer array in a subroutine, using

```
MASK(I)=2**(NBITS-I)
```

When a new list is being created the double sum over molecules is done in the usual way, but with a little extra code.

```
DO 1 I=1,N-1
  IWORD=1+(I-1)/NBITS
  IBIT=MOD(I-1,NBITS)+1
  DO 1 J=I+1,N
    (Calculate the distance, and jump to 1 if out of range)
    JWORD=1+(J-1)/NBITS
    JBIT=MOD(J-1,NBITS)+1
    LIST(IWORD,J)=LIST(IWORD,J).OR.MASK(IBIT)
    LIST(JWORD,I)=LIST(JWORD,I).OR.MASK(JBIT)
1  CONTINUE
```

The logical `.OR.` operator generates a word which has a 1 in each bit position which was non-zero in either of the operands; it would also set to 1 any position which was 1 in both operands, but that situation cannot arise here. As a result, the `NWORDS` for molecule 1 contain a list of `N` bits with 1's in the locations of those molecules within the cut-off distance. The interaction of `I` with itself is set as out of range, by default, and it may be necessary to change this when internal degrees of freedom are considered. The lists are easy to use, as shown here.

```
(choose a molecule, I, to be moved, and move it)
(begin the calculation of the change in energy)
  JWORD=1
  JBIT=0
  DO 3 J=1,N
  JBIT=JBIT+1
  IF(JBIT.LE.NBITS) GO TO 4
  JWORD=JWORD+1
  JBIT=1
  4 IF(.NOT.(LIST(JWORD,I).AND.MASK(JBIT))) GO TO 3
(Test for the true potential cutoff, etc.)
  CONTINUE
```

The logical `.AND.` generates a word containing 1's only in those positions which are non-zero in both of the operands; because the mask has a single non-zero bit, this constitutes a test for the corresponding bit in the list. In this test, if the bit in the list is zero the logical product results in a zero word (`.FALSE.`), and the `.NOT.` operator converts it to `.TRUE.`, so that the `GO TO` is executed. The success of this code depends on the compiler recognizing as 'true' a word that has a single non-zero bit somewhere within it; the standard IBM compilers do this, as do most others, but it is worth testing in case a slight modification of the code is needed with a particular compiler. Because the operations are logical, this method runs very quickly, as well as being compact. The integer divide and `MOD` operations used here for clarity can be replaced by simple counting, and this usually results in slightly faster execution.

The `MD` form is simpler because the double sum over molecules enumerates the (complete) list of pairs in the same sequence each time, so that no bookkeeping information need be stored. The list is simply a string of $N*(N-1)/2$ bits, and is written and tested in much the same way as for the `MC` case.

```
KWORD=1
KBIT=0
DO 10 I=1,N-1
DO 10 J=I+1,N
KBIT=KBIT+1
```

```

    IF(KBIT.LE.NBITS) GO TO 11
    KWORD=KWORD+1
    KBIT=1
  11 CONTINUE
  (if this pair is not active jump to 10)
    LIST(KWORD)=LIST(KWORD).OR.MASK(KBIT)
  10 CONTINUE

```

Again, the logical .OR. operation is used to insert a “1” in the appropriate position in the list to flag an active pair. The use of the list is equally simple.

```

    KWORD=1
    KBIT=0
    DO 12 I=1,N-1
    DO 12 J=1+1,N
    KBIT=KBIT+1
    IF(KBIT.LE.NBITS) GO TO 13
    KWORD=KWORD+1
    KBIT=1
  13 CONTINUE
    IF(.NOT.(LIST(KWORD).AND.MASK(KBIT))) GO TO 12
  (this is an active pair)
  12 CONTINUE

```

The logic here exactly parallels that described before.

The storage saved depends on the system size and the range of interaction, as well as on the word length. For 256 molecules, with the cut-off at half the box length, and using 16-bit integers, the MC form uses 16x256 words, and the MD form 8x256, compared with around 128x256 and 64x256, respectively. For larger system sizes the savings are correspondingly greater, and it is with larger systems that the extra effort in coding this version becomes not only worthwhile, but necessary. However, the range of interaction can have a profound influence on the situation. For very large systems having only short ranged interactions the average number of neighbours may be less than $N/NBITS$, and the conventional technique then becomes more efficient. If the system size is very large, even comparatively short neighbour lists per particle can prove troublesome, and the logical testing of every pair is inefficient. Then, it may be worth using the cell index method. Each edge of the box is divided into an integer number of pieces of equal length, the pieces being at least as long as the radius of interaction. Each coordinate is assigned an integer index based on the subsection in which it falls, so that $3N$ words of storage are required. In testing, only molecules in the same or next-neighbour boxes can be in range. By testing each coordinate in turn, so the vast majority of possible neighbours are rejected at the first test etc., a relatively efficient algorithm with minimal storage requirements is obtained. One final word of warning deals with dilute systems.

Even though the expected average number of neighbours may be small, condensation can change that very dramatically. The method described here gives the correct result, whatever the state of the system. In the conventional form the storage required for the lists increases with the number of interacting neighbours, and this can lead to problems in a system which can condense.

For modest sized systems with an interaction range which is a substantial fraction of the box length, the binary form of the neighbour lists is compact and efficient. As the system size increases, so does the advantage of the binary form, as long as the ratio of the range of interaction to box length does not get too small.

References

- [1] S. M. Thompson, CCP5 Newsletter, 8, p20 (March, 1983)
- [2] L. Verlet, Phys. Rev. 159, 98 (1967)