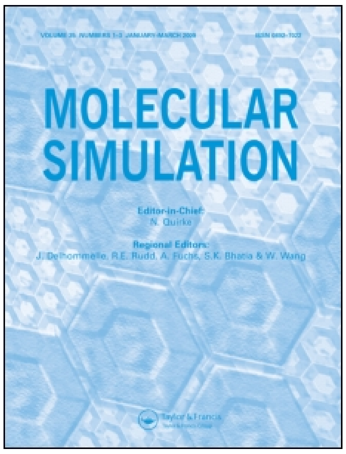

Fast Coding of the Minimum Image Convention

M. Hloucha[a]; U. K. Deiters[a]
[a] Institut für Physikalische Chemie, Universität zu Kölin, Köln, Germany

Molecular Simulation, 1998, Vol. 20, pp. 239–244
Reprints available directly from the publisher
Photocopying permitted by license only

FAST CODING OF THE MINIMUM IMAGE CONVENTION

M. HLOUCHA and U. K. DEITERS

Institut für Physikalische Chemie, Universität zu Köln, Luxemburger Str. 116, D-50939 Köln, Germany

(Received March 1997; Accepted April 1997)

Various algorithms for the implementation of the minimum image convention are compared. On many platforms algorithms with if statements are most efficient.

Keywords: Minimum image convention; algorithms

1. INTRODUCTION

Because of their huge requirements of computer resources, Monte Carlo or molecular dynamics simulations of real liquids are usually carried out with rather small numbers of molecules. For simulations of macroscopic properties, typically 250–1000 molecules are placed into a cubic simulation box. In order to minimize surface effects, periodic boundary conditions are introduced [1], i.e. the simulation box is surrounded by an infinite number of replicas. Consequently, pair interactions must be calculated between the closest periodic images of two molecules. This method, called minimum image convention, was first used by Metropolis *et al.* [2].

Even with the most powerful computers available today, simulations of polar and polarizable molecular fluids require several days of computing time [3]. In such simulations, most of the total CPU time is spent for the calculation of electric fields and interaction energies, and – within these routines – for the calculation of distances.

We compared several algorithms for the calculation of distances with periodic boundary conditions to obtain the optimum performance on different processing systems.

2. THEORY

In the following we consider a cubic simulation box of length B with $a = B/2$, i.e. the boundaries of the original box are at $-a$ and a. The relation between the coordinates of one particle in the box, x_B, and its image in the nth periodic box, x, is given by

$$x = x_B + nB \tag{1}$$

where n is an integer number.

The distance Δx between two particles i and j anywhere in the periodic lattice is given by

$$\Delta x = x_i - x_j = x_{B,i} - x_{B,j} + B(n_i - n_j) \tag{2}$$

The problem is now, according to the minimum image convention, to find the smallest possible distance between the particles or any of their images

$$\Delta_{\min} x = \Delta x - kB \tag{3}$$

Evidently, the desired distance is the remainder of the division of Δx by B:

$$\Delta_{\min} x = \mathrm{rem}(\Delta x, B) = \Delta x - B\,\mathrm{nint}\left(\frac{\Delta x}{B}\right) \tag{4}$$

Here nint denotes the "nearest integer".

If x_i and x_j belong to the same simulation box replica, the smallest particle distance is obtained by the following corrections:

$$\begin{aligned} \Delta x < -a &\Rightarrow \Delta_{\min} x = \Delta x + B \\ \Delta x > +a &\Rightarrow \Delta_{\min} x = \Delta x - B \end{aligned} \tag{5}$$

Direct implementation of this method leads to an algorithm with two *if* statements, which we refer to as Algorithm 0 and sketch below in C-style notation. Correct results are obtained for $-1.5B < \Delta x < 1.5B$.

Algorithm 0 Basic algorithm with two *if* clauses, based on Equation (5).

```
if $(dx > a)\, dx = dx - B$;

else if $(dx < -a)\, dx = dx + B$;
```

The two Equations (5) can be combined into single one, in which there is only one *if* clause:

$$|\Delta x| \geqslant 1 \Rightarrow \Delta_{\min} x = \Delta x - B\,\mathrm{sign}(\Delta x) \tag{6}$$

While Fortran contains the sign-transfer function, which can be used here well, the extraction of the sign is less easy in C. Algorithm 1 illustrates a possible solution.

Algorithm 1 Algorithm with one *if* clause (Equation (6)).

```
int k;

k = ((dx < 0.0) << 1) - 1;

if (fabs(dx) > a) dx = dx + k*B;
```

Let $\Delta\tilde{x} = \Delta x/a$ denote a normalized distance. Then Equation (5) can be transformed as follows:

$$\Delta x_{\min} = \begin{cases} \Delta x + (+1)B & \Delta\tilde{x} \leqslant -1 \\ \Delta x + (0)B & -1 < \Delta\tilde{x} < +1 \\ \Delta x + (-1)B & \Delta\tilde{x} \geqslant +1 \end{cases}$$

$$= \Delta x - B\,\mathrm{int}(\Delta\tilde{x}) \tag{7}$$

Here int denotes the standard rounding function of C or Fortran (rounding towards zero, truncation), which is automatically invoked when a floating point number is stored to an integer target (integer type casting). The realisation of this method is shown as Algorithm 2, which does no longer contain any *if* statements.

Algorithm 2 Using one type casting, provided that $-B < \Delta x < B$.

```
int k;

k = dx * (1/a);

dx = dx - k * B;
```

Algorithm 2 may fail, however, if the two particles under consideration are further apart than one box length – in the case of molecules with site-site potentials, this can even happen, if the centers of the molecules are in the same simulation box. But it is possible to extend this algorithm to the range $-1.5B < \Delta x < +1.5B$ by applying type casting twice (Algorithm 3).

Algorithm 3 Using two type castings, provided that $-1.5B < \Delta x < 1.5B$.

```
int k;

k = dx * (1/a);

dx = dx - k * B;

k = dx * (1/a);

dx = dx - k * B;
```

Finally, a different correction method can be constructed from Equations (2–3) by noting that Δx_{min} is, by necessity, insensitive to changes of Δx by multiples of the box length. Hence it is possible to shift Δx to positive values by adding a sufficiently large offset, thus avoiding all difficulties arising from its sign, i.e. if $n_i - n_j$ in Equation (2) is an integer between $-m$ and $+m$, we relocate the interacting pair of particles in the area between 0 and $2mB$. For positive arguments however, the rounding and truncation functions are related by

$$\operatorname{nint}(x) = \operatorname{int}\left(x + \frac{1}{2}\right) \tag{8}$$

Hence the resulting correction is

$$\Delta x_{\text{min}} = (\Delta x + mB) - B\left(\operatorname{int}\left(\frac{\Delta x + mB}{B}\right) + \frac{1}{2}\right) \tag{9}$$

For typical simulation situations (molecules not longer than half the box size), $m = 3$ is a reasonable choice; the resulting Algorithm 4 produces correct results for $-(m + 0.5)B < \Delta x < \infty$.

Algorithm 4 Offset by integer multiples of the box length

```
int $m = 3$;

int $k$;

$dx = dx * (1/B) + m$;

$k = dx + 0.5$;

$dx = dx - B * k$;
```

3. RESULTS AND DISCUSSION

The algorithms for all fast computation of distances were tested for their performance on RISC as well as vector processors. For this purpose a C program was written in which five distance values, of which two needed the periodic boundary correction, were treated 2×10^7 times by the analyzing algorithm. The execution times for the algorithms on the different computer systems are listed in Table I.

It turns out that on the IBM, HP, SUN and DEC RISC platforms, algorithms with *if* statements are most efficient. The highly variable execution times for Algorithm 1 are caused by the necessary sign checking, which had to

TABLE I CPU times for 1×10^8 minimum image corrections using different algorithms on different processing systems. Note that Algorithm 2 has a smaller argument range than the others

manufacturer	*IBM*	*HP*	*HP*	*Sun*	*DEC*	*SGI*	*Cray*
system:	*RS6000/ 990*	*712–80*	*735–99*	*Ultrasparc*	*AXP500*	*SC900*	*T916/ 12512*
optimization:	*–O3*	*+O3*	*+O4*	*–xO3*	*–O3*	*–O2*	*–O3*
Algorithm:							
0	32.8	27.8	23.5	12.8	11.1	40.9	21.3
1	24.0	95.6	30.6	38.9	16.7	33.7	10.8
2	38.8	30.8	21.5	15.3	15.4	8.6	2.9
3	85.2	51.0	33.8	29.2	34.5	14.9	3.0
4	40.6	37.2	25.3	30.1	21.0	10.7	3.1

be programmed explicitly because of the lack of an appropriate function in the programming language.

On SGI and Cray vector processors drastic speed increases can be achieved by using algorithms with type castings. The reason for this performance enhancement is due to the fact that only loops which do not contain *if* statements can be vectorized.

In a former publication [4] a considerable speed-up had been obtained by invoking the remainder function as suggested by Equation (4). However, this was due to the fact that the remainder operation was hard-coded in the CPU (Motorola 68020/68881); using the remainder function from the C math library on a modern HP RISC workstation slowed it down by a factor of 6!

In our Monte Carlo fluid simulation program [3], a subprogram generating a list of all atom-atom distances using the Algorithm 3 reached a computational speed of 990 MFlops on the Cray T90 processor, which has a theoretical maximum of 1200 MFlops. Conversely we must conclude that an ill-adapted algorithm can bring down even a Cray supercomputer to the performance level of a 5,000 $ workstation.

Acknowledgments

We are grateful to the Jülich Supercomputing Center and the Cologne Regional Computing Center (RRZK) for providing the necessary computing time.

The authors would like to thank J. Gehm and T. Bönniger (RRZK) for helpful discussions.

References

[1] Allen, M. P. and Tildesley, D. (1987). *Computer Simulation of Liquids*. Clarendon Press, Oxford.
[2] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H. and Teller, E. (1953). *J. Chem. Phys.*, **21**, 1087–1092.
[3] Hloucha, M. and Deiters, U. K. (1997). *Mol. Phys.*, **90**, 593–597.
[4] Deiters, U. K. (1989). *Molecular Simulation*, **3**, 343–344.