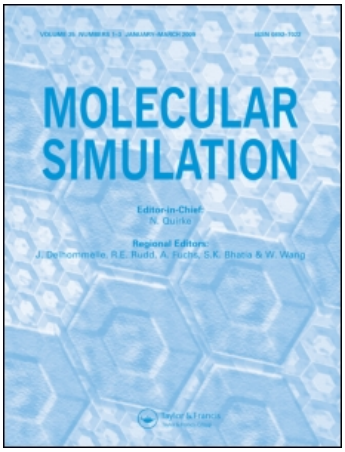


Molecular Simulation
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713644482

Microcomputers Against Supercomputers? On Geometric Partition of the Computational Box for Vectorized MD Algorithms

Jacek Kitowski[a]; Jacek Mośiński[a]
[a] Institute of Computer Science, Cracow, Poland

To cite this Article Kitowski, Jacek and Mośiński, Jacek(1992) 'Microcomputers Against Supercomputers? On Geometric Partition of the Computational Box for Vectorized MD Algorithms', Molecular Simulation, 8: 3, 305 — 319
To link to this Article: DOI: 10.1080/08927029208022485
URL: http://dx.doi.org/10.1080/08927029208022485

Molecular Simulation, 1992, Vol. 8, pp. 305–319
Reprints available directly from the publisher
Photocopying permitted by license only

MICROCOMPUTERS AGAINST SUPERCOMPUTERS? ON GEOMETRIC PARTITION OF THE COMPUTATIONAL BOX FOR VECTORIZED MD ALGORITHMS*

JACEK KITOWSKI and JACEK MOŚCIŃSKI

Institute of Computer Science, AGH al. Mickiewicza 30, 30-059 Cracow, Poland

(*Received September 1990, accepted January 1991*)

In the paper three vector algorithms (with original data structures and no nearest neighbour lists) for Molecular Dynamics simulation are compared and their timings presented. The timings have been obtained on ETA 10-P*108 and IBM 3090/150E (with Vector Facility) computers as well as on several microprocessors (e.g. MOTOROLA MC68020/68881, INTEL i80386/387, i486 and INMOS TRANSPUTER T800).

KEY WORDS: Molecular dynamics, Lennard-Jones potentials, computational algorithms, data structures for vector processing, microprocessors.

1 INTRODUCTION

The Molecular Dynamics (MD) method is a well established and widely used technique in many fields of computing. This includes 3-D simulations of bulk properties (e.g. [1]–[4]) as well as 2-D simulations of microscale hydrodynamics [5]. Calculating forces (potentials) acting on each particle is the most CPU-time consuming part of simulation. It uses 95 + % of the computing time so optimization of the method must be directed to this calculation.

Supercomputers are very powerful tools for modern science. To exploit their advantages, suitable algorithms are needed, since efficiency of any MD program critically depends on data structures. Automatic tools (e.g. vectorizers, optimizers, etc.) are not capable of improving the program efficiency significantly, if the data structures are not properly chosen for vector processing. On the other hand due to modern technology new microprocessors come into market. In some particular circumstances they could be effectively used for simulation purposes at small fraction of cost in comparison with supercomputers.

The purpose of the paper is twofold. First, we present timings of three algorithms for short-range MD (3-D) simulation, which are suitable for vector processing facilities of modern supercomputers. The timings have been obtained on ETA 10-P*108 and IBM 3090/150E (with Vector Facility) computers. Second, we compare the timings for supercomputers with those for some modern microprocessors (i.e.

*Extended version of the poster presented at the SERC CCP5 Meeting "Architecture and Algorithms in Condensed Phase Simulations", St. Andrews, Scotland, 2–5 July, 1990.

MC68020/68881, i80386/387, T800 and i80486), which have been benchmarked with similar algorithms.

The algorithms do not use a table of nearest neighbours, although algorithms with the nearest neighbour list are the most efficient at present and approach the technological limit of the computers. The main drawback is the high operating memory demand for list keeping. In this paper original geometric partitions of the computational box are proposed and discussed.

In the algorithms interparticle forces are described by the 12/6 Lennard-Jones (L-J) short-range potential. The Newtonian equations of motion are solved with the leap-frog scheme. In most cases look-up tables are not applied and the interparticle forces as well as the potential energy are directly computed with the *force* loop.

In Section 2 the main ideas of the algorithms are shown which were partly described previously in [6]. Section 3 contains simulation results. Conclusions are presented in Section 4.

2 VECTOR ALGORITHMS

2.1 Pipe Method

The first algorithm concerns simulation of dynamics of particles confined in a long, narrow cylinder, so that it is called the *pipe method.* It could be used for studies of thermodynamics and transport properties of fluids in micropores (e.g. [7]). The ratio of the cylinder radius, R_W, to L-J parameters, σ, is limited to $R_W/\sigma = 2 \div 6$.

The full description of the method is presented in [8]. In the paper the main features are repeated since the *pipe method* is a basis for other algorithms development. Periodic boundary conditions (*pbc*) are introduced along the cylinder axis only in z-direction). For simplicity, interactions between particles and the cylindrical wall are described by the L-J potential also. The algorithm is based on the observation, that in fluid of uniform density the number of neighbours interacting with a given atom is nearly constant. Hence on a base of the cutoff radius, R_C, the integer cutoff number, *NCUT*, is introduced, where *NCUT* is a number of neighbours interacting potentially with a given atom. Although *NCUT* depends on geometry of the system, assumed potential and density, for the geometry under study *NCUT* is defined as a number of particles contained in a slice of the computational cylinder of height equal to $R_C + \Delta R_C$, ΔR_C represent an extra thickness introduced due to local fluctuations of fluid density. The value of ΔR_C (and *NCUT* consequently) should be carefully chosen, since the larger *NCUT* value the longer execution time is observed. On the other hand too small *NCUT* value might cause particles losing from the potential cutoff.

Particles in the cylinder are sorted due to their Z coordinates (forming ZS vector) and the index vector is set up to return to original particles indices. X and Y particles coordinates are gathered due to the index vector, which results in XS and YS vectors determination. For interparticle forces calculation (i.e. calculation of distances between particles) the computational cylinder is stepwise shifted in respect to its copy to subsequent neighbouring particles from 1 to *NCUT*. Using this algorithm the forces calculations are fully vectorized.

The *NCUT* application leads to the following algorithm for the *FORCES* routine (FORTRAN 200 notation is used).

1. Sort Z coordinates (in increasing order) along the cylinder axis into the sorted

vector ZS

$$ZS(1) \leqslant ZS(2) \leqslant \ldots \leqslant ZS\,(NPARTS - 1) \leqslant ZS(NPARTS), \quad (1)$$

where $NPARTS$ is number of particles in the cylinder, and set up the index vector used to return to original particles indices.

2. Gather X and Y particles coordinates into XS and YS vectors due to the index vector.
3. In order to handle the *pbc* in z direction, extend XS, YS and ZS vectors to length $NPARTS + NCUT$

$$\begin{aligned} &NPARTS1 = NPARTS + 1 \\ &XS\,(NPARTS1;\, NCUT) = XS(1;\, NCUT) \\ &YS\,(NPARTS1;\, NCUT) = YS(1;\, NCUT) \\ &ZS(NPARTS1;\, NCUT) = ZS(1;\, NCUT) + HCYL \end{aligned} \quad (2)$$

The cylinder height is represented by $HCYL$.

4. Compute distances between neighbouring particles and forces acting on the particles

DO $JJ = 2,\ NCUT + 1$

{compute distances between particles}

$$\begin{aligned} &DX\,(1;\, NPARTS) = XS(1;\, NPARTS) - XS(JJ;\, NPARTS) \\ &DY(1;\, NPARTS) = YS(1;\, NPARTS) - YS(JJ;\, NPARTS) \\ &DZ(1;\, NPARTS) = ZS(1;\, NPARTS) - ZS(JJ;\, NPARTS) \end{aligned} \quad (3)$$

.....

{compute pair-forces acting between particles}

{compute total forces and reactions on particles}

.....

ENDDO

These calculations may be interpreted as distances computations between particles from the original cylinder and particles from a copy of the cylinder shifted in z direction. Components of forces acting on sorted particles are obtained in a completely vectorized way with maximal possible vector length equal to $NPARTS$.

5. Compute particle-wall interactions.
6. Scatter the calculated forces to return to the original indices.

The algorithm is programmed in FORTRAN 200 language. For simplicity look-up tables are not used and interparticle forces are directly computed within the loop. The chosen $NCUT$ and ΔR_C values imply that sometimes distances between particles are longer than the assumed R_C value. Therefore the vectorized instruction WHERE is used in order to set the reciprocal square distances between particles to zero in that case.

In the algorithm a high height/radius ratio for the cylinder makes possible applications of long vectors suitable for ETA 10-P or CYBER-205 computers.

2.2 *Stick Method*

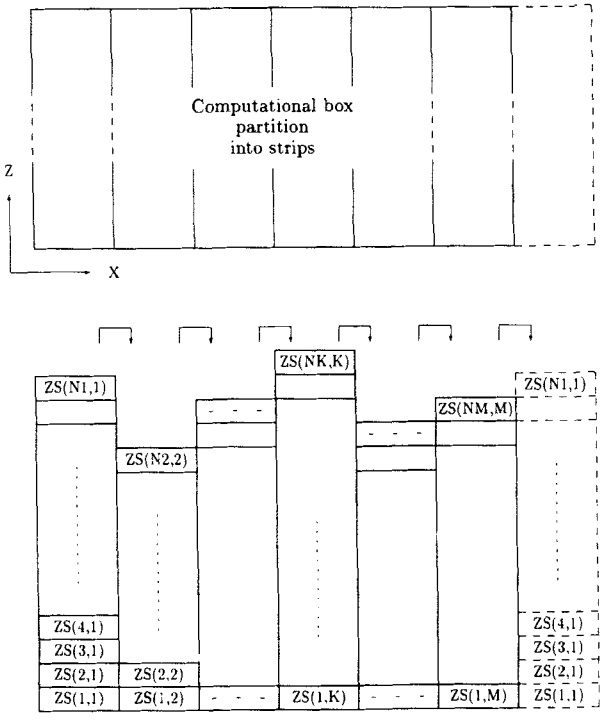
The second algorithm is an extension of the *pipe* method to 3-D bulk systems with classical *pbc*. To calculate mutual interactions with large degree of vectorization, data are organized in a special way. The computational box is divided in $(x-y)$ plane into the elongated in z direction box columns (called *sticks*, Figure 1). Hence the method is called the *stick method*. In 2-D simulation the box is divided into *strips*, thus the first version of the method has been called the *strip* one [6].

The *pbc* conditions in z direction are treated in a similar way than in the *pipe method*, while in the $(x-y)$ plane — as in the *link-cell* method.

The particles in every stick are sorted (in increasing order) due to their Z coordinates, so the vectors ZS are determined (see Figure 1). As in the previous algorithm the integer cutoff number $NCUT$ is used. The forces calculations are performed in two main steps: (a) calculate forces between particles in the same stick — the method used for the long cylinder is adopted, (b) calculate forces between particles from adjacent sticks (mentioned by arrows in Figure 1). For the (b) step one stick is successively shifted in respect to its neighbour (cf. Figure 2). Since the numbers of particles in sticks are a little bit different, the vector alignment is necessary.

There are two versions of the method:

- *basic version.* In this version the integer cutoff number, $NCUT$, being a simula-



$$ZS(1,K) \leq \ldots \leq ZS(NK,K), \quad \text{for} \quad K = 1,\ldots,M$$

Figure 1 Data structure for the *stick method*.

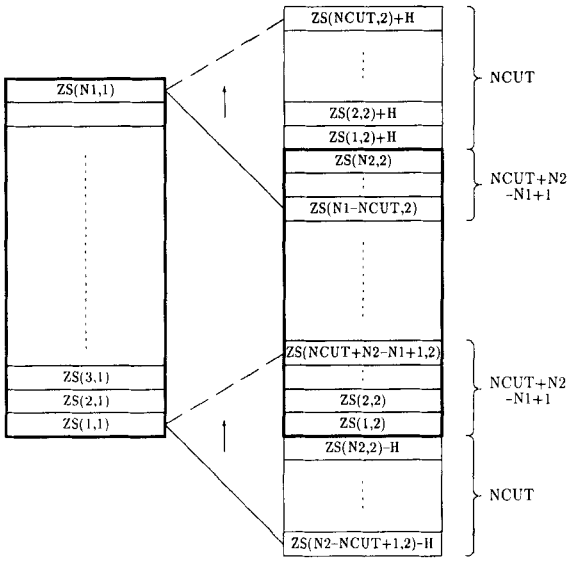


Figure 2 Vector calculations of differences in particle coordinates for neighbour sticks ($K = 1,2$), with particle numbers $N1 > N2$ ($N1 = NIP(1)$, $N2 = NIP(2)$). H represents the box height.

tion parameter, is defined prior to the time-loop and kept constant during the simulation. The same *NCUT* value is used for every stick pair regardless the number of particles in the sticks,

- *local, dynamic cutoff number version.* Two cutoff numbers are introduced. First of them, *NCUTIS*, is used for interactions computation between particles from the same stick. It is a simulation parameter kept constant during the simulation. The second one, *NCUT*, is used for determination of interactions between particles from adjacent sticks. It is determined locally for each stick pair according to the difference in particles belonging to the pair sticks. The average value, *NCUTS*, of integer cutoff number of local stick pair, *NCUT*, is smaller in this version than in the previous one, hence the shorter execution time is observed.

The algorithm is summarized as follows:

1. Divide the computational box in *NC* sticks of width $\geqslant R_C$ and set number of particles in every stick $NIP(K)$ ($K = 0, 1,....,NC - 1$). (In Figure 1 *NC* is called *M*.) Numbers of sticks in x and y directions are equal to *NCX* and *NCY* respectively. In the present version $NCX = NCY$.
2. Sort the particles in every stick along z axis in ascending sequence of Z coordinates values

$$ZS(1,K) \leqslant ZS(2,K) \leqslant ... \leqslant ZS(NIP(K), K) \tag{4}$$

and set up a table of original indices for every K stick.
3. Gather X and Y particles coordinates into XS and YS vectors according to the table of original indices.
4. To handle *pbc* along the stick (in z direction) set up the extended vectors ZS,

XS and YS for every K stick

$$
\begin{aligned}
&NIPK = NIP(K)\\
&\text{DO } II = 1,NCUT\\
&\quad ZS(NIPK + II,K) = ZS(II,K) + HST\\
&\quad ZS(-NCUT + II,K) = ZS(NIPK - NCUT + 1,K) - HST\\
&\quad XS(NIPK + II,K) = XS(II,K)\\
&\quad XS(-NCUT + II,K) = XS(NIPK - NCUT + 1,K)\\
&\quad YS(NIPK + II,K) = YS(II, K)\\
&\quad YS(-NCUT + II,K) = YS(NIPK - NCUT + 1,K)\\
&\text{ENDDO}
\end{aligned}
\tag{5}
$$

The box height is represented by HST. In scheme (5) applied in the *local, dynamic cutoff number version* $NCUT$ is the maximal number from along those calculated for every pair of sticks.

5. Calculate forces acting on all particles.

a) calculate forces betwen particles in the same K stick — the *pipe method* is adopted.

b) calculate forces between particles from adjacent sticks. This is done for every K stick ($K = 0,...,NC - 1$) and its JC neighbour ($JC = 0,...3$, since in the 3-D case, due to symmetry, each stick as 4 neighbouring sticks used for forces evaluation). Prior to forces calculations the relative positions of neighbouring sticks, JC, are mapped into positions in the computation box, $JCELL$. Additionally *pbc* are taken into account also

$$
\begin{aligned}
&\text{DO } II = 1,NIPK\\
&\quad XIS(II) = XS(II,K) - ZCONVX\\
&\quad YIS(II) = YS(II,K) - ZCONVY\\
&\text{ENDDO}
\end{aligned}
\tag{6}
$$

$XCONVX$ and $ZCONVY$ represent *pbc* in x and y directions respectively.

Calculate forces and reactions

— *basic version.* The integer cutoff number is fixed during the simulation. Since the numbers of particles in the sticks are different, three cases are applied $NIP(K) = NIP(JCELL)$, $NIP(K) > NIP(JCELL)$ and $NIP(K) < NIP(JCELL)$. The forces computations differ slightly in the above cases. As an example, the scheme for distances and forces calculations is presented for K and $JCELL$ sticks pair ($NIPK = NIP(K)$, $NIPJ = NIP(JCELL)$) (while $NIPK > NIPJ$.

$$
\begin{aligned}
&J3 = 0\\
&NCUT1 = -NCUT + 1\\
&NCUTJH = NCUT - NIPK + 1\\
&\text{DO } JJ = NCUT1, NCUTJH + NIPJ
\end{aligned}
$$

```
J3 = J3 + 1
J2 = NCUT + 1 - J3
J1 = JJ - 1
DO II = 1,NIPK
  DZ = ZS(II,K) - ZS(J1 + II,JCELL)
  DY = YIS(II) - YS(J1 + II,JCELL)
  DX = XIS(II) - XS(J1 + II,JCELL)                    (7)
  DIST2 = 1./(DX*DX + DY*DY + DZ*DZ)
  IF (DIST2.LT.RRCUT2) DIST2 = 0
  ......
  {calculate forces}
  ......
ENDDO
......
{calculate reactions}
......
ENDDO
```

For numerical purposes $RRCUT2 = 1./R_C^2$.

— *local, dynamic cutoff number version.* It is similar to scheme (7) apart from application of the locally determined *NCUT*, which is used for each pair of sticks separately. It is computed with

$$NCUT = NCUTI + |\, NIPK - NIPJ \,|, \tag{8}$$

where $|\, NIPK - NIPJ \,|$ is the absolute value of difference of particle numbers from *K* and *JCELL* sticks and *NCUTI* is defined prior to the time-loop. In this version the following modifications are also applied (similar to suggestions in [9]).

- look-up tables for potential and pair-forces,
- *FORCES* routine is split into smaller parts,
- sticks updating is done every *NSTUPD* = 1, 10, 20) number of time-steps.

6. Return to original particles indices.

2.3 Bulk Shift Method

In the third algorithm particles of the 3-D cubic system are mapped into a computational grid (or into cells) in such a way that every grid point is occupied by only one particle at most (i.e. in every cell there is at most one particle). The particles coordinates are placed in auxiliary tables *XG*, *YG* and *ZG*, whose indices represent 3-D grid coordinates similarly to *Monotonic Logical Grid algorithm* ([10]). The algorithm could be also treated as a vectorization of the method of Quentrac and Brot [11].

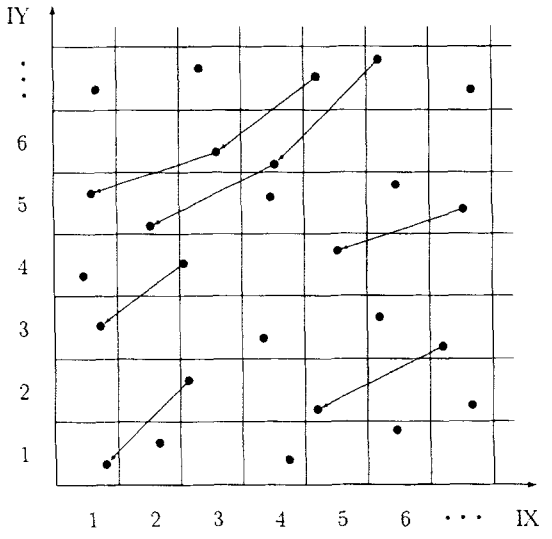


Figure 3 Scheme for the *bulk shift* method (2-D case). The forces for all pairs of particles separated by the grid distance (*JJ*,*JK*) are calculated (on the picture *JJ* = 2 and *JK* = 1).

In the algorithm *NCUT* is applied as well, which maps R_C into number of neighbouring grid points in each direction. The base grid is extended with *NCUT* points in x, y and z directions.

Assume, that *JC* is a number of a chosen neighbour of every particle. To calculate the forces acting on the particles, the forces between every particle and its *JC* neighbour are evaluated (see Figure 3 for 2-D case as an example). This can be interpreted by having the grid and its extended copy shifted by the grid distance resulting from *JC* so the method is called the *bulk shift method.*

Similar to the *link-cell method* the half of neighbours of each particle, *NNC*, is considered for forces evaluations only ($NNC = ((2 \times NCUT + 1)^3 - 1)/2$).

Due to local fluctuations in particle density some grid points might not be occupied by the particles. In order to calculate forces in a vector way *artificial ("ghost") particles* are introduced for that purpose. Their coordinates are chosen in such a way that the interactions: empty grid point-particle and empty grid point-empty grid point are eliminated from pair-forces calculations.

The algorithm is summarized as follows:

1. Construct the 3-D grid
   a) set pseudo-random initial values for arrays *XG*, *YG* and *ZG* from $(0, 10^{33})$ in order to eliminate interactions with empty grid points. (The upper limit of pseudo-random number is arbitrary.) Set initial values (equal to 0) for *LG* table used for return to original indices.
   b) generate grid tables *XG*, *YG* and *ZG*

```
DO J = 1,NPARTS
  IX = IFIX (SFX*X(J) + 1.)
  IY = IFIX (SFY*Y(J) + 1.)
  IZ = IFIX (SFZ*Z(J) + 1.)
```

$$XG(IX,IY,IZ) \tag{9}$$
$$YG(IX,IY,IZ) = Y(J)$$
$$ZG(IX,IY,IZ) = Z(J)$$
$$LG(IX,IY,IZ) = J$$

ENDDO

SFX, *SFY* and *SFZ* are scaling factors for automatic increasing of number of grid points to achieving one particle at every grid point at most. For this criterion *LG* table is used as well. In the present version of the algorithm $SFX = SFY = SFZ$ due to the cubical computational box.

c) if the grid is not properly chosen (the grid points are occupied by more than one particle) then the number of grid points (cells) in every direction, *NCX*, *NCY* and *NCZ*, is increased (due to the cubic computation box $NCX = NCY = NCZ$)

$$NCX = NCX + 1$$
$$NCY = NCX \tag{10}$$
$$NCZ = NCX$$

the scaling factor and the integer runoff number are modified

$$SFX = \text{FLOAT}\,(NCX)/RNCK$$
$$\text{NCUT} = \text{IFIX}\,(SFX^{*}RCUT) + 1 \tag{11}$$

and the grid generation is repeated.

2. Set extended grid using *pbc* condition for

x direction: $-NCUT + 1, \ldots, 0, \ldots, NCX, \ldots, NCX + NCUT$,

y direction: $-NCUT + 1, \ldots, 0, \ldots, NCY, \ldots, NCY + NCUT$,

z direction: $1, \ldots, NCZ, \ldots, NCZ + NCUT$.

3. Perform the bulk shift subsequently to all *JC* neighbours (for $JC = 1$ to *NNC*)
a) calculate integer coordinates of the *JC* neighbour of every particle in respect to the original grid point. These coordinates are kept in *NEIGHX*, *NEIGHY*, and *NEIGHZ* tables respectively, which are generated at the begining of simulation

$$JJ = NEIGHX\,(JC)$$
$$JK = NEIGHY\,(JC) \tag{12}$$
$$JL = NEIGHZ\,(JC)$$

b) calculate forces acting on all particles due to the bulk shifted to the *JC* neighbour (the base grid is shifted in respect to its copy by the vector (*JJ*, *JK*, *JL*)).

DO $IZ = 1, NCZ$
  DO $IY = 1, NCY$
    DO $IX = 1, NCX$
      $DX = XG(IX, IY, IZ) - XG(IX + JJ, IY + JK, IZ + JL)$
      $DY = YG(IX, IY, IZ) - YG(IX + JJ, IY + JK, IZ + JL)$

```
DZ = ZG(IX, IY, IZ) − ZG(IX + JJ, IY + JK, IZ + JL)
DIST2 = 1./(DX * DX + DY * DY + DZ * DZ)          (13)
IF (DIST2.LT. RRCUT2) DIST2 = 0.
.......
{calculate forces}
.......
   ENDDO
ENDDO
ENDDO
```

Similar to the *stick method* $RRCUT2 = 1./R_C^2$

4. Calculate reaction forces and return to original indices.

3 SIMULATION RESULTS

Simulation results for the described algorithms have been performed for argon with $R_C = 2.5\sigma$ and temperature $T = 86.5K$. The time-step for integration of the Newtonian equations of motion is equal to $\Delta t = 10^{-14}$ s. The applied computers are: ETA 10-P*108 (with 21 ns clock) and IBM 3090/150E with Vector Facility (with 17.2 ns clock). On ETA 10-P the Fortran programs are compiled with the assistance of the Pacific-Sierra VAST-2 preprocessor.

In the methods different kinds of sorting algorithms are used

- *RADIXSORT* for ETA 10-P ([12,13]),
- *DSORTX* from *ESSL* library for IBM 3090,
- *QUICKSORT* for other machines.

In most cases average timings have been obtained for 3000 time-steps (for the *pipe method*) and for 100 time-steps (for other methods).

Results for the *pipe method* (*MP*) are partially presented in [8]. They are obtained for the reduced density, $\rho^* = \rho\sigma^3$, equal to $\rho^* = 0.68$, and for the number of particles in sphere with R_C radius (i.e. for nearest neighbours), η, equal to $\eta = 44.5$ for ETA 10-P and for $\rho^* = 0.76$ ($\eta = 49.7$) for other computers. The cylinder radius (in σ units) $R_W/\sigma = 2$. For *NCUT* determination a parameter, $\omega = \Delta R_C/R_C$, is applied, which is equal to $\omega = 0.3$.

The *stick method* (*MS*) is tested for $\rho^* = 0.72$ ($\eta = 47.1$). For the *basic version* $\omega = 1.75$. For the *local dynamic cutoff number version* there are two input parameters: $\omega_l = \Delta R_{CS}/R_C$ and $\omega = \Delta R_C/R_C$. First of them is used for *NCUTIS* determination, while the second – for *NCUTI* and, according to Equation (8), for local *NCUT*. In the simulation $\omega_l = 0.5$ and $\omega = 0.75$ are adopted. No *vector directives* are applied on IBM 3090 with Vector Facility.

In Figure 4 two timing results for the *stick method* (*local, dynamic cutoff number version*) are presented. The execution time, τ, is calculated per time-step and particle and *NPARTS* means the number of particles in the comptational box. In the case a) *global VAST directives* are neglected, while in the case b) the following ones are applied: ASSERT DEPENDENCY (IGNORE) and ASSERT VECTOR (HIGH). The difference between cases a) and b) is $\sim 15\%$. The average value of the local integer cutoff number, *NCUTS*, is approximately independent of the number of particles, hence the algorithm is $o(NPARTS)$.

The *bulk shift method* (*BS*) is tested for $\rho^* = 0.72$ ($\eta = 47.1$). On ETA 10-P *two*

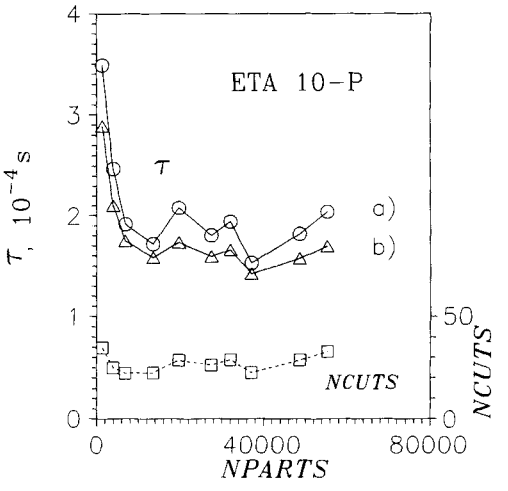


Figure 4 Execution time and average cutoff number for the *stick method* (*MS*/20). a) without *global VAST directives*, b) with *global VAST directives*.

global VAST directives are used: ASSERT DEPENDENCY (IGNORE) and ASSERT VECTOR (HIGH). On IBM 3090 with Vactor Facility the *vector directive* C*VDIR:IGNORE RECRDEPS (...) is applied.

In Figure 5 a comparison of the *MP*, *MS*/20 (*local dynamic integer cutoff version* with *NSTUPD* = 20) and *BS* methods is presented. In each case efficiency increase with the vector length is observed, however for greater number of particles τ is approximately constant, thus the algorithms are $o(NPARTS)$. Discontinuous characteristics for the *BS* method results from decreasing number of time-steps in simulation for greater number of particles, so that the grid is enlarged to smaller extent in respect to slighter particle density fluctuations.

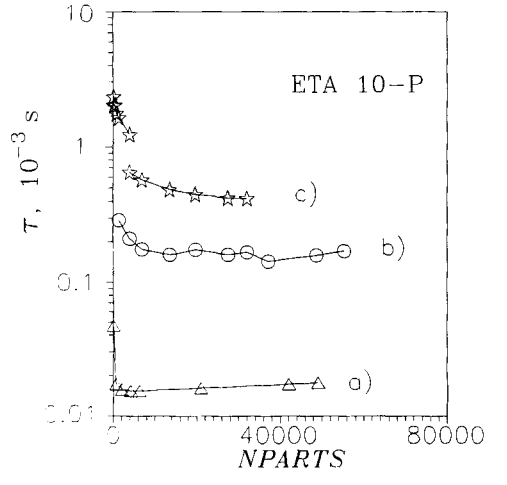


Figure 5 Comparison of the execution time for different algorithms. a) *pipe method*, b) *stick method* (*MS*/20), c) *bulk shift method*.

Differences in timings for the algorithms amount to one order of magnitude, in respect to different data structures. This confirms fundamental role of proper data structure for vector processing.

Following the Bakker's statement ([14]): *"Although supercomputers have opened ways to perform large-scale computer experiments, only a small group of physicists and chemists use them as a standard facility due to their costs and availability"*, the algorithms have been tested on series of microprocessors also and the timings are reported. In addition a comparison with the sequentially optimized *link-cell method* (*LC* - [15]) is presented. The last algorithm is programmed in the C language and tested for argon-krypton mixture with $\eta = 55$.

The average timings are summarized in Table 1.
In Table 1 the following notation is used:

DSI-020 - MC68020/68881 (12.5 MHz),
AT-386 - i80386/387 (20 MHz),
T800 - TRANSPUTER T800 (20 MHz),
AT-486 - i486 (25 MHz),
MP - *the pipe method*,
MS - *the stick method* (*basic version*),
MS/20 - *the stick method* (*local, dynamic cutoff number version*) with sticks update every 20 time-step,
BS - the *bulk shift method*,
LC - the *sequential link-cell method* - C language,
([1]), ([2]) - see explanation in the text,
([3]) - Turbo C (v2.0) 16-bit compiler.

Timings for the microcomputers are obtained using the algorithms (and programs) developed for vector supercomputers - thus they are not tuned for sequential realization. There are two exceptions mentioned by ([1]) and ([2]). In both an "if" statement is used to avoid forces calculations unless the distance between particles is equal to or less than R_C. Additional "if" statement is applied in ([2]) case to avoid evaluation of forces for empty grid points. For the microcomputers 32-bit compilers have been used except ([3]) case.

4 CONCLUSIONS

It is worthwhile to compare the obtained calculation speed (or timings) with those for other methods. For this purpose, the performance speed, p, is introduced, which

Table 1 Timings comparison

Computer	τ [10^{-4} s/step/particle]				
	MP	*MS*	*MS/20*	*BS*	*LC*
DSI-020	86.[1]	-	-	-	220.
AT-386	28.[1]	461.	328.	168.[2]	130.[3]
T800	13.[1]	-	178.	-	35.
AT-486	6.3[1]	-	82.	-	26.[3]
IBM 3090/VF	-	5.5	~3.	25.0	-
ETA 10-P	0.16	3.1	1.6	18.3	-

represents the number of particles moved per cpu-sec. For the proposed algorithms, values of the performance speed for supercomputers are presented in Table 2. They concern typical simulation conditions with η values mentioned in Section 3 (case A) and estimation for $\eta = 33.5$ nearest neighbours (case B). The last case is presented in order to compare the obtained results with other methods.

Several methods are used for the comparison (see Table 3). The standard *vector link-cell method* (*VLC*) based on [16,17] has been coded in Fortran 77 and run on ETA 10-P (with $\rho^* = 0.72$ and $\eta = 47.1$). The performance speed for other methods is taken from the literature (and rescaled if necessary, to ETA 10-P speed and $\eta = 33.5$). Obviously the rescaling gives very rough comparison only.

The timings for the *link-cell method* with enhanced vectorisation by loop re-ordering (*VLCE*) are given in [3] for the two-pipe Cyber 205, half-precision and $R_C = 2^{1/6}\sigma$.

The first version of the *vector pyramid method* (*VPYR*1) is described in reference [18]; its timings concern exactly ETA 10-P and $\eta = 33.5$.

The rest of the data from Table 3 relates to the Rapaport *layers method* (*MLY*) (given in [3] for the two-pipe Cyber 205, half-precision and $R_C = 2^{1/6}\sigma$), the third version of the *vector pyramid method* (*VPYR*3) [9,18] and the *slab method* (*SLAB*) [9,18]. *VPYR*3 and *SLAB* have been tested on ETA 10-P for $\eta = 33.5$.

The performance speed of a simple test version of the *monotonic logical grid method* (*MLG*) are reported in [10] for Cray X-MP/12 and $\eta = 124$. Rescaling to ETA 10-P speed and $\eta = 33.5$ gives $p = 45000$. This test version could be informally compared with calculation of interactions (forces) in methods with nearest neighbour lists (neglecting their construction); in this case $p = 50000$ for the *SLAB* algorithm [9].

Timings for the first three methods shown in Table 3 are similar, since the methods do not use any nearest neighbour list. In the rest the list and other acceleration techniques are applied.

From the developed algorithms the *pipe method* (*MP*) is the most efficient one, since the highly unsymmetrical, non-typical computational box, suitable for simulation of (micro-) pore properties. The vector length is equal to the number of particles in the box. Its performance speed, p, exceeds the speed for highly optimized algorithms with the nearest neighbour list (see Table 3).

For the 3-D bulk simulation the speed of the *stick method* (*MS*/20) is ~ 4 times higher than for the *VLC* method (on ETA 10-P) and about twice higher than the speed of the *VPYR*1 algorithm (Table 3). It is, however, $2 \div 4$ times slower than the algorithms with the neighbour list mentioned above.

Table 2 Performance speed for the developed algorithms. A) for simulation conditions, B) estimation for $\eta = 33.5$.

Computer	*p, [particles/s]*			
	MP	*MS*	*MS/20*	*BS*
A)				
IBM 3090/VF	–	1800.	3300.	400.
ETA 10-P	62500.	3250.	6100.	550.
B)				
IBM 3090/VF	–	2530.	4600.	560.
ETA 10-P	83000.	4570.	8550.	770.

Table 3 Comparison of performance speed for different algorithms (literature data are rescaled to ETA 10-P speed and $\eta = 33.5$).

Computer	*p, [particles/s]*					
	VLC	*VLCE*	*VPYRI*	*MLY*	*VPYR3*	*SLAB*
IBM 3090/VF	3200.	–	–	–	–	–
ETA 10-P	2200.	3000.	3800.	20000.	29000.	35000.

The *bulk shift method* is inefficient, in respect to nested loops and short vectors generated. Its efficiency could be probably higher using the future FORTRAN 8x standard. From tests it follows also, that using this approach for the 2-D case, better efficiency is observed [19].

For supercomputing purposes algorithms have to be well tuned to the computer architecture. Otherwise, it may happen that highly optimized sequential algorithms used on microcomputers have similar timings to the vector ones. The first example is presented in Table 1 (compare the *LC* and *BS* methods). The second example concerns similar timings obtained for the *vector link-cell* method (on ETA 10-P) and for the highly optimized *scalar piramid* algorithm with the nearest neighbour list (tested on T800 – [18]). Even if the timings are different (cf. Table 1 for *LC* method on AT-486 with 16-bit compiler and *MS*/20 on ETA 10-P) use of microprocessors could be profitable.

So that – is it always necessary to use expensive vector computers for real scientific simulations? Maybe the advent of new microprocessors (e.g. i860 with vector processing, TRANSPUTER H1, etc.) will partially change old minds?

Acknowledgements

The authors wish to thank Dr. Z.A. Rycerz from the University of Western Ontario, London, Canada, Dr. M. Bargiel, Dr. W. Dzwinel, Dr. Z. Skotniczny and Dr. M. Bubak from AGH, Poland for many valuable discussions, suggestions and assistance during the work. Many thanks are due to Professor P.W.M. Jacobs, director of the Center of Chemical Physics, UWO, Canada for computer time on ETA 10-P, arranging the fellowship for J.M. and many helpful discussions. We are grateful to Project P/04/192/90-2 for partial financial support.

References

[1] F.F. Abraham, "Computational statistical mechanics. Methodology, applications and supercomputing", *Advances in Physics*, **35** (1986) 1.

[2] D.W. Heermann, "*Computer Simulation Methods in Theoretical Physics*", Springer Verlag, Berlin (1986).

[3] D.C. Rapaport, "Large-scale molecular dynamics simulation using vector and parallel computers", *Comput. Phys. Reports*, **9**, 1(1988) 3.

[4] P.W.M. Jacobs, Z.A. Rycerz and J. Mościński, "Molecular dynamics of ionic crystals", *Adv. Solid State Chem.*, in press.

[5] D.C. Rapaport, "Microscale hydrodynamics: Discrete-particle simulation of evolving flow patterns", *Phys. Rev. A* **36**, 7 (1987) 3288.

[6] J. Mościński, M. Bargiel, J. Kitowski, Z., Skotniczny, Z.A. Rycerz and P.W.M. Jacobs, "Vectorized molecular dynamics algorithms for very large number of particles", *Proc. of 1989 International Conference on Supercomputing*, June 5–9, 1989 Crete, Greece (ACM Conference Proceedings).

[7] U. Heinbuch and J. Fischer, "Liquid argon in cylindrical carbon pore: molecular dynamics and Born-Green-Yvon results", *Chem. Phys. Lett.*, **135** 6 (1987) 587.

[8] J. Mościński, J. Kitowski, Z.A. Rycerz and P.W.M. Jacobs. "A vectorized algorithm on the ETA 10-P for molecular dynamics simulation of large number of particles confined in a long cylinder", *Comput. Phys. Commun.*, **54** (1989) 47.

[9] Z.A. Rycerz and P.W.M. Jacobs, "On the efficiency of vectorized molecular dynamics algorithms of order N", *J. Comput. Phys.*, in press.

[10] J., Boris, "A vectorized near neighbours algorithm of order N using a Monotonic Logical Grid", *J. Computat. Phys.*, **66** (1986) 1.

[11] B. Quentrac and C. Brot, "New method for searching for neighbours in molecular dynamics calculations", *J. Comput. Phys.*, **13** (1975) 430.

[12] J. Krieger, "Sorting on Cyber 205" in: *Proc. of the 1983 Conference on CYBER 205 in Bochum*, eds. H. Ehlich and K.-H. Schlosser, Bochumer Schriften zur Parallel Datenverarbeitung 4 (Rechenzentrum der Ruhr-Universitaet Bochum, Dezember 1984).

[13] J. Mościński, Z.A. Rycerz and P.W.M. Jacobs, "Timing results of some internal sorting algorithms on the ETA 10-P", *Parallel Computing*, **11** (1989) 117.

[14] A.F. Bakker, "Dedicated parallel computers for molecular simulations", *Abstracts of a joint meeting of the Collaborative Computational Project 5 of the SERC and Institute of Physics: "Architecture and Algorithms in Condensed Phase Simulations"*, St. Andrews, 2–5 July 1990.

[15] M. Bargiel, W. Dzwinel, J. Kitowski and J. Mościński, "C-language molecular dynamics program for the simulation of Lennard-Jones particles", *Comput. Phys. Commun.*, in press.

[16] D.M. Heyes and W. Smith "CRAY vectorized link-cell code", *CCP5 Information Quarterly for MD and MC Simulations*, **26** (1987) 68, informal Newsletter (Daresbury Laboratory, Warrington, England).

[17] D.M. Heyes, "Correction to CRAY vectorised link-cell code", *CCP5 Information Quarterly for Computer Simulation of Condensed Phases*, **28** (1988) 63, informal Newsletter (Daresbury Laboratory, Warrington, England).

[18] Z.A. Rycerz and P.W.M. Jacobs, "A molecular dynamics algorithm of order N", *J. Comput. Phys.*, in press.

[19] J. Kitowski, "Vector algorithms for molecular dynamics simulation of large number of particles", *Scientific Bulletins of the Stanislaw Staszic University of Mining and Metallurgy – Automatics*, vol. 53, Cracow, Poland (1990), in polish.