DARESBURY LABORATORY

INFORMATION QUARTERLY for COMPUTER SIMULATION OF CONDENSED PHASES

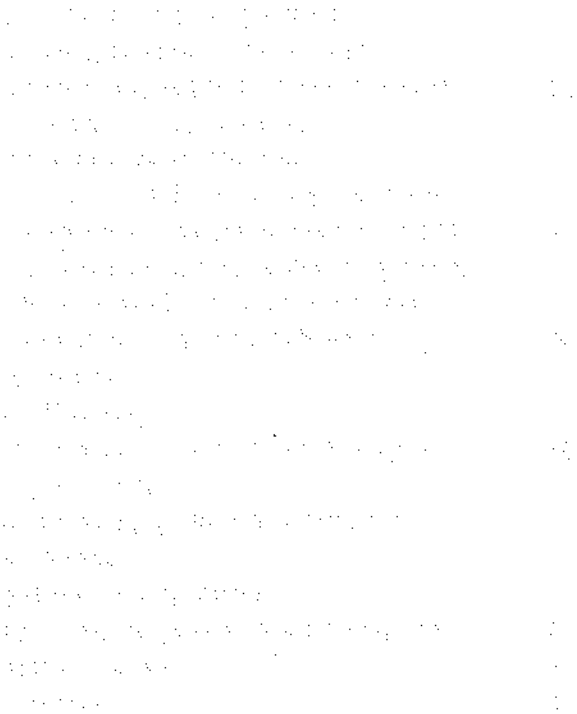
An informal Newsletter associated with Collaborative Computational Project No. 5 on Molecular Dynamics, Monte Carlo and Lattice Simulations of Condensed Phases.

Number 27 March 1988

Contents

Editor: Dr. William Smith
Deputy Editor: Dr. Maurice Leslie

Science & Engineering Research Council
Daresbury Laboratory, Daresbury,
Warrington WA4 4AD, England.



Editorial

CCP5 has now submitted its renewal proposal to SERC for a four year rolling grant. Themes included in the renewal are exploitation of new hardware, development of new computational algorithms, simulation of fluids with application to colloids and rheology, simulation of macromolecules, simulation of materials, quantum simulations and industrial applications. We should hear in the near future whether this has been successful.

Contributors to the current issue.

Our thanks go to:

D. Fincham	Computer Centre, University of Keele, Keele, Staffs ST5 5BG.
J Mościński W Dzwinel J Kitowski M. Bargieł	Institute of Computer Science AGH, 30-059 Kraków, Poland.
L. Woodcock	School of Chemical Engineering, University of Bradford, Bradford BD7 1DP.
W. Smith	Theory and Computational Science Division, S.E.R.C. Daresbury Laboratory, Daresbury, Warrington WA4 4AD, Cheshire.

General News

DISCUSSION MEETING Parallel Computers and Molecular Simulation

University of Keele. April 6-7 1988

This is planned to be a small informal meeting, limited to around 25 participants, to exchange experiences and ideas about the use of parallel computers in simulations. It is also hoped to have presentations by several manufacturers. CCP5 may be able to pay expenses of participants. If you would like to attend, please write or Email to David Fincham, Computer Centre, University of Keele, Keele, Staffordshire ST5 5BG, UK [Fincham @ UK.AC.KL.GEC] giving your name, address and the topic on which you would like to give a short presentation.

NATO ASI NATO Advanced Studies Institute. Computer Simulation of Fluids, Polymers and Solids

The NATO Scientific Affairs Division have approved a proposal submitted by the CCP5 executive committee for an ASI in the field of computer simulation of Condensed Matter. The school will be held at the University of Bath from 4th - 17th Sept 1988 and will present a general survey of the techniques and application of computer simulation methods. Although the school will be of an advanced tutorial nature, there will be an opportunity to present new results. It is hoped to be able to provide financial assistance for participants, especially for graduate students. Further information may be obtained from Prof. C. R. A. Catlow, Department of Chemistry, University of Keele, Keele, Staffordshire, ST5 5BG, U.K.

MEETING ON BIO-ORGANIC APPLICATIONS CCP5 is to organise a meeting on the subject of bio-organic applications of computer simulation, which provisionally, is scheduled for Easter 1988. More details of this will be revealed as the organisation develops.

UMRCC In October UMRCC announced full screen access to ROSCOE via VM/CMS. The Peer Review scheme has now been operational on the Cyber 205 for four months. All resources consumed on the 205 are accounted to an approved project. This includes all category S jobs. Once all the resources of a project have been used, category S jobs are used to allow the project to continue computing provided there are machine resources available.

ULCC Phoenix 3 is now the recommended logon procedure. Users of Phoenix 1 should consult the October 1987 newsletter for details of how to change. There are a number of changes to command syntax and a number of commands no longer available. On the CRAY, from January 1988 stricter COS PASSWORD control has been enforced. This will not affect users who submit jobs from their own accounts. However, if you run jobs on the CRAY using another account number, you will need to supply a password. Also, the default dataset access mode has been changed from read to no access. This means that other users who use your datasets will no longer be able to do so unless you take the necessary action. (See November newsletter for details.) An upgrade of the COS operating system and CFT compiler to level 1.15 is still scheduled for the near future.

RAL A CRAY user group meeting was held at Rutherford Laboratory on 15th October 1987. Some of the points which emerged during discussion are

1. The operating system has recently been upgraded to COS 1.16. This will allow better accounting for microtasking. Also, jobs using the solid state storage device will be able to share this resource. In the very long term RAL will change to the UNIX based unicos operating system.
2. RAL will be obtaining the FORGE software package on a months trial basis soon and would like to hear from anyone interested in using it. The package aids vectorisation of scalar code.
3. Users who use more than 3 Mwords of memory in a single job should consider microtasking, so their jobs do not degrade machine performance.
4. A number of new options are now, or shortly will be available to carry out graphics. Readers are referred to the December issue of Arclight for more details.

CRAY TIME CCP5 participants are reminded that CCP5 has an annual allocation of Cray time at each of the centres: London (Cray 1s) and Rutherford (Cray XMP-48), which is available for the development of simulation programs prior to a grant allocation. At present CCP5 is allocated 15 hours a year at Rutherford with a minimal allocation at ULCC. Readers who wish to use some of this allocation should write to the CCP5 Secretary, Dr. M. Leslie, TCS Division, SERC Daresbury Laboratory, Daresbury, Warrington WA4 4AD.

THE CCP5 PROGRAM LIBRARY.

ADMIXT	[MD,LJA/MIX,LF,TH+MSD+RDF]	W. Smith
CARLOS	[MC,VS+Aquo,TH]	B. Jonsson S. Romano
CARLAN	[DA,CARLOS structure analysis]	B. Jonsson S. Romano
CASCADE	[LS,DIL,EM,TH+STR]	M. Leslie/ W. Smith
CURDEN	[DA,Current Density Correlations]	W. Smith
DENCOR	[DA,Density Correlations]	W. Smith
HLJ1	[MD,LJA,LF,TH+MSD+RDF]	D.M. Heyes
HLJ2	[MD,LJA,LF,TH+MSD+RDF+VACF]	D.M. Heyes
HLJ3	[MD,LJA,LF/LC,TH+MSD+RDF]	D.M. Heyes
HLJ4	[MD,LJA,LF/CP+CT,TH+MSD+RDF]	D.M. Heyes
HLJ5	[MD,LJA/SF,LF,TH+MSD+RDF]	D.M. Heyes
HLJ6	[MD,LJA,TA,TH+MSD+RDF]	D.M. Heyes
HMDIAT	[MD,LJD,G5+Q4,TH+MSD+QC]	S.M. Thompson
HSTOCH	[MD/SD,VS+BA,LF+CA,TH]	W.F. van Gunsteren/ D.M. Heyes
MCN	[MC,LJA,TH]	N. Corbin
MCLSU	[MC,LJA,TH]	C.P. Williams/ S. Gupta
MCRPM	[MC,RPE,TH+RDF]	D.M. Heyes
MDATOM	[MD,LJA,G5,TH+RDF+MSD+QC]	S.M. Thompson
MDATOM	[MD,LJA,LF,TH+MSD+RDF]	D. Fincham
MDDIAT	[MD,LJD,LF+CA,TH+MSD]	D. Fincham
MDDIATQ	[MD,LJD+PQ,LF+CA,TH+MSD]	D. Fincham
MDIONS	[MD,BHM,LF,TH+MSD+RDF+STF]	D. Fincham/ N. Anastasiou
MDLIN	[MD,LJL,G5+Q4,TH+MSD+QC]	S.M. Thompson.
MDLINQ	[MD,LJL+PQ,G5+Q4,TH+MSD+QC]	S.M. Thompson
MDMANY	[MD,LJS+FC,LF+QF,TH]	D. Fincham/ W. Smith
MDMIXT	[MD,LJS/MIX,LF+QF,TH]	W. Smith
MDMPOL	[MD,LJS+FC/MIX,LF+QF,TH]	W. Smith/ D. Fincham
MDPOLY	[MD,LJS,G5+Q4,TH+MSD+QC]	S.M. Thompson
MDMULP	[MD,LJS+PD+PQ/MIX,LF+QF,TH]	W. Smith
MDSGWP	[MD,LJA/SGWP,LF,TH+VACF+RDF+QC]	W. Smith/ K. Singer
MDTETRA	[MD,LJT,G5+Q4,TH+MSD+QC]	S.M. Thompson
MDZOID	[MD,GAU,LF+QF,TH+MSD+RDF+VACF]	W. Smith
SCN	[MC,LJA,RFD,TH]	N. Corbin
SURF	[MD,BHM/TF/2D,LF,TH+RDF]	D.M. Heyes
SYMLAT	[LS,PIL,EM+SYM,TH+STR]	Harwell

THBFIT	[LS,PIL,EM,Potential fitting]	Harwell
THBPHON	[LS,PIL/3B,EM,Phonon dispersion]	Harwell
THBREL	[LS,PIL,EM,TH+STR]	Harwell

Key:

Program types:	MD	Molecular dynamics
	MC	Monte Carlo
	LS	Lattice simulations
	SD	Stochastic dynamics
	DA	Data analysis
System models:	LJA	Lennard-Jones atoms
	LJD	Lennard-Jones diatomic molecules
	LJL	Lennard-Jones linear molecules
	LJT	Lennard-Jones tetrahedral molecules
	LJS	Lennard-Jones site molecules
	RPE	Restricted primitive electrolyte
	BHM	Born-Huggins-Meyer ionics
	SGWP	Spherical gaussian wavepackets
	TF	Tosi-Fumi ionics
	VS	Variable site-site model
	BA	Bond angle model
	PD	Point dipole model
	PQ	Point quadrupole model
	MIX	Mixtures of molecules
	GAU	Gaussian molecule model
	FC	Fractional charge model
	PIL	Perfect ionic lattice model
	DIL	Defective ionic lattice model
	3B	3-body force model
	2D	Two dimensional simulation
	SF	Shifted force potential
	FC	Fractional charge model
Algorithm:	G5	Gear 5th order predictor-corrector
	Q4	Quaternion plus 4th. order Gear P-C.
	LF	Leapfrog (Verlet)
	QF	Quaternion plus Fincham algorithm
	LC	Link-cells MD algorithm
	CP	Constant pressure
	CT	Constant temperature
	TA	Toxvaerd MD algorithm
	CA	Constraint algorithm
	EM	Energy minimisation
	SYM	Symmetry adapted algorithm
	RFD	Rossky-Friedman-Doll algorithm

Properties:	TH	Thermodynamic properties.
	MSD	Mean-square-displacement
	RDF	Radial distribution function
	STF	Structure factor
	VACF	Velocity autocorrelation function
	QC	Quantum corrections
	STR	Lattice stresses

University of Manchester
Institute of Science and Technology

Research on the Simulation of Chain Molecules

Two Post-Doctoral Research Assistantships are available for molecular dynamics computer simulation research in the following areas

(1) Glass transition phenomena in linear chain polymers.
This project is sponsored by ICI plc who are likely to have a direct future interest in dynamic modelling of these systems.

(2) Structural and dynamic properties of monolayer and multilayer films of chain molecules (Langmuir-Blodgett films). This project is funded by the SERC. There is strong current interest in such materials for their electronic applications.

Both positions are available for up to two years at a starting salary in the range £9 000 - £11 000 p.a. plus benefits, dependent on qualifications and experience. Excellent computing facilities are available, and you will be joining a lively research group concerned with diverse applications of molecular dynamics.

Applications enclosing a curriculum vitae should be sent as soon as possible to Dr. J.H.R. Clarke, Chemistry Department, U.M.I.S.T., Manchester M60 1QD.

The Visit of Professor Parrinello to Daresbury Laboratory

W. Smith

2 November 1987

The visit of Professor M. Parrinello to Daresbury aroused considerable interest. Visitors were drawn from Manchester, Keele and Oxford, to complement the attendance of Daresbury associated staff at the seminar given by him. The subject about which he had been asked to speak was the novel method devised by himself and Roberto Car at Trieste to determine the electronic and structural properties of the element silicon. The method is essentially a synthesis of molecular dynamics, simulated annealing and density functional theory.

Recognising that molecular dynamicists and band-structure theoreticians tend to come from separate camps, Professor Parrinello began by describing the features of both formalisms. In molecular dynamics one is attempting to solve the problem of motion in a many-particle system. Given that one knows the nature of the interaction between the particles (defined by the potential energy function) one can construct a Lagrangian from which the equations of motion may be derived and solved numerically using any of the standard MD techniques. In conventional MD the nature of the potential is assumed *a priori* and most often, empirical pair potentials are used. Since the potentials are quantum mechanical in origin, there has always existed the possibility of calculating the true potential at each timestep in the molecular dynamics, though this has always been regarded as too expensive to contemplate.

A molecular dynamics scheme in which the forces are calculated by quantum mechanics was outlined by Professor Parrinello, and it involved the use of density functional theory. Starting with the Hohenberg-Kohn theory he described the derivation of the Kohn - Sham equations, which are self-consistent single-particle equations which can be used iteratively to determine the energies of electronic states. The equations contain an exchange-correlation term which may be calculated using the local density approximation. In principle these equations may be solved at each time step of a molecular dynamics algorithm - for each new configuration of atoms - and the forces computed directly. One would then have a molecular dynamics method with no empirical parameters whatsoever. Unfortunately however the iterative solution of the Kohn-Sham equations is too computationally expensive to provide a practical scheme.

Professor Parrinello's solution to this problem was characteristically innovative. He set up a classical Lagrangian for the system which in-

cluded a pseudo-kinetic energy term for the time derivative of the wavefunction (i.e. $\frac{1}{2}\mu|\dot{\phi}_i|^2$, where μ is a fictitious mass for the wavefunction) and also the orthogonality constraints for the single particle wavefunctions. This allowed the derivation of pseudo-classical equations of motion for the system, which could be solved by molecular dynamics. The net kinetic energy of this extended system, including both the pseudo-kinetic energy of the wavefunction and the classical kinetic energy of parent ions, provides a definition of the system temperature, which may be used in a simulated annealing experiment. Gradual reduction of the temperature in the course of the simulation is equivalent to a minimisation of the system energy and is formally equivalent to a direct solution of the self-consistent Kohn-Sham equations.

Not only does this method provide an elegant way of determining the minimum energy, but it may also be used to give the dynamics of the ions on the Born-Oppenheimer surface. If the mass μ is made sufficiently small that the adiabatic principle is obeyed, diagonalisation of the Kohn-Sham equations is rendered unnecessary and the simplest possible dynamical description of the ions on the surface is obtained. Thus the objective of providing a simulation scheme free of empirical parameters is achieved. In fact the new method is superior in other respects. It is possible by this method to excite selected vibrational modes, but in contrast with the standard solution, damping of the vibration (due to accumulated inaccuracies in the standard method) does not arise.

In collaboration with Roberto Car at Trieste, Professor Parrinello has studied many aspects of crystalline, liquid and amorphous silicon. Their model system consisted of 54 silicon atoms in a cubic, constant volume system with the usual periodic boundary conditions. The simulations showed that the crystal is 4-coordinated, but the liquid is nearer 6-coordinated. The calculated mean- square-displacement agreed with experiment. Amorphous silicon was produced by cooling the liquid at a rate of 10^{14} to 10^{15} degrees per second. The radial distribution function produced compared very well with the density functional theory result. The coordination number peaked at 4, but the spread ranged from 3 to 5. An analysis of the bond-angle distributions showed an average of 109 degrees in the amorphous form, and six-fold rings of silicon atoms are dominant.

The Fourier transform of the velocity autocorrelation function provides the phonon density-of-states, which showed a good comparison with the experimental result. The differences that arose were explained by the rather short cut-off used in the simulations (6 Å). The calculated electronic density-of-states of the liquid was essentially free-electron like but in the amorphous form a band gap was seen to open up at 4eV.

The work of Pantelides has indicated that the amorphous form contains 3- and 5- coordinated silicon atoms. These defects were also found in the simulations of Parrinello and Car. These observations were used to explain the specific heat anomaly in amorphous silicon.

In conclusion the method was shown to reproduce structural, dynam-

ical and electronic properties in good agreement with experiment. The possibility of extending the method to study other systems was clealy available. The limitations of the method concern the size of the simulated system and the time scales that can be examined. The description of the electronic states is less complete than that given by other methods and the validity of the dynamical simulations hinges in turn on the validity of the adiabatic approximation.

Professor Parrinello's talk was followed by a short account by W. Smith of his work (with J.H.R. Clarke, UMIST) in attempting to model the phases of crystalline potassium nitrate by the molecular dynamics method of Parrinello and Rahman. The chosen pair potentials were of the Buckingham exp/6 type, with fractional charges on the nitrate anion, which was treated as a rigid, planar triangle with four sites. The model revealed several distinct phases under different conditions of temperature and pressure, but comparison with real potassium nitrate was not good. Nevertheless, the observed phases tended to show some of the features of the expected phases. The points of similarity and the nature of the differences were described by Dr. Smith. He concluded by outlining some of the difficulties of using this method to study phase properties (potentials aside), in particular, the problem of identifying new phases as they occur in the course of the simulation.

Mean-Field Stokesian Dynamics

Les Woodcock

December 21, 1987

The use of steady-state NEMD simulations to explore non-linear flow phenomena is now well-established but,so far, quantitative predictions for experimental non-Newtonian fluids have not been possible[1]. The difficulty rests with the requirement to establish steady-state shear conditions. In most computations modified Newtonian dynamics have been deployed to fix the particle kinetic energies[2,3] whereas in reality the mean particulate velocities are shear-rate dependent in the non-linear region when the boundary conditions are constant.

In steady-state flow the boundary-driven shearing force puts heat into the system according to

$$Q = \eta\dot{\gamma}^2 \tag{1}$$

where Q is the power input per unit volume, η is the viscosity and $\dot{\gamma}$ is the velocity gradient (rate of strain). In isokinetic simulations the total (particle) kinetic energy, which is defined as

$$E = \frac{m}{2}\sum_i [v_y^2 + v_z^2 + (v_x - \dot{\gamma}Y)^2] \tag{2}$$

is held constant by scaling the peculiar velocities (i.e. the particle velocities relative to the flow field) according to

$$\Delta\underline{v} = \underline{v}(1-f) : f = (E/E_o)^{1/2} \tag{3}$$

as frequently as permissible by the algorithm being used.

Whilst this procedure may not correspond to the practicalities of shearing atomic or molecular liquids at ultra-high frequencies, it is closely related to the shear mechanisms in colloidal suspensions where f, the mean damping variable in modified Newtonian dynamics , takes the rôle of the mean hydrodynamic friction constant. By considering hard-sphere model dynamics we can establish the connection between the Stokes friction constant and the scaling of the kinetic energies, osmotic pressures including the stress, and diffusivities obtained for isokinetic conditions.

Consider the equations-of-motion of N Stokesian spheres in a Newtonian fluid medium of viscosity η_m, mass m, diameter σ, in volume V with particle positions $\underline{r}_i$ and velocities $\underline{v}_i$ etc. Neglecting many-body contributions to the hydrodynamic friction force, i.e. taking only a one-body "mean-field" term, the Stokes friction constant (C) determines the

equations-of-motion;

$$\frac{d^2\underline{r}_i}{dt^2} = -C\frac{d\underline{r}_i}{dt} : C = \frac{3\pi\sigma\eta_m}{m} \tag{4}$$

whereupon C^{-1} sets the time scale.

On solving equation (4)

$$\underline{r}_i(t) = \underline{r}_i(0) + \underline{v}_i(0)\left(\frac{1-\exp -Ct}{C}\right) \tag{5}$$

noting that on expanding the exponential and taking the limit $C \to 0$ Newtonian dynamics are recovered.

To calculate the "next collision time" between two spheres, the time t is determined by the condidtion

$$\underline{r}_{ij}^2(t) - \sigma^2 = 0 \tag{6}$$

substituting for $\underline{r}_{ij}(t)$ from equation (5) gives the quadratic equation

$$\underline{r}_{ij}^2(0) - \sigma^2 + 2\underline{r}_{ij}(0)\underline{v}_{ij}(0)\left(\frac{1-\exp -Ct}{C}\right) + \underline{v}_{ij}^2(0)\left(\frac{1-\exp -Ct}{C}\right)^2 = 0 \tag{7}$$

putting

$$x = \left(\frac{1-\exp -Ct}{C}\right)$$

then, as with classical spheres, x is obtained from the positive real root of the general quadratic and the collision time t is obtained from

$$t = -\frac{1}{C}\ln(1-Cx) \tag{8}$$

The equation-of-motion for the velocities is given by the time-derivative of equation (5)

$$\underline{v}_i(t) = \underline{v}_i(0)\exp -Ct \tag{9}$$

At every collision in a dynamical simulation all the particle positions and velocities are recalculated according to the equations-of-motion (5) and (9). It should be noted that in this simple scheme, for shear flow, the flow field velocity can only be influential at the driving boundaries. When a Y-dependent component of the v_i^x is included the quadratic equation (7) becomes a quartic, which is, however, still amenable to straightforward computational solution. The osmotic pressure tensor is obtained, just as in Newtonian dynamics, by taking the time-average of the momentum transferred over all the collisions.

The time scaling relationship between Newtonian dynamics and mean-field Stokesian dynamics shows that the two methods have the same dynamical properties in the limit of a sufficiently high scaling frequency in the damped Newtonian case, every collision time-step say. Thus, if all the velocities are multiplied by the same scaling factor, defined in equation (9), the two methods become equivalent. In continuous potential dynamics, e.g. soft spheres, momentum scaling at every time step

effects a similar correspondence. Whereas in previous damped Newtonian NEMD computations the kinetic energy is constant and the friction coefficient varies, in mean-field Stokesian dynamics the reduced friction coefficient $C/\dot{\gamma}$ is the constant of the calculation and the mean (hydrodynamic) kinetic energy varies.

Mean-field stokesian dynamics for hard-spheres can lend itself to the scaling of the list of times to the next collision, a trick that enables fast computation for hard spheres since only those spheres involved in the "last collision" need to have the collision times recalculated completely. For all spheres not involved the list x_i is updated according to

$$x_i(new) = x_i(old) - \Delta x \tag{10}$$

where

$$C\Delta x = \exp(\Delta t - t_i(old)) - \exp(-t_i(old)) \tag{11}$$

in practice the expansion can be truncated to ease the computation

$$x_i(new) = x_i(old) - \Delta t - \Delta t^2/2 - \Delta t \ln(1 - x_i(old)) \tag{12}$$

If the particle velocities are updated at every collision to take account of the changed flow-field velocity component, this time-saving listing device is not applicable.

Calculations using the above schemes are in progress and will be reported in due course, but, in the meantime, the scaling variables of mean-field Stokesian dynamics for hard spheres can be used to examine experimental scaling laws. NEMD studies of the HS model for homogeneous shear flow have been reported by Erpenbeck[4] and Naitoh and Ono[5]. In both computations, the kinetic energy was allowed to increase with time so that the reduced shear rate in the usual HS-units was effectively decreasing with time. Naitoh and Ono[5] were able to deduce the isokinetic reduced flow curve data by rescaling the kinetic energy and shear-rate. We can use these hard-sphere scaling laws to make predictions about the general rheology of colloidal suspensions.

The corresponding state scaling laws for the hard-sphere model in the isokinetic form and Stokesian form are as follows.

Reduced Properties	Isokinetic	Stokesian
time	$t^\dagger = t(E_o/m\sigma^2)^{1/2}$	$t^* = tC$
(for thermal systems	$E_o = kT$)	
shear rate	$\dot{\gamma}^\dagger = \dot{\gamma}(m\sigma^2/E_o)^{1/2}$	$\dot{\gamma}^* = \dot{\gamma}/C$
viscosity	$\eta^\dagger = \eta\sigma^2/(mE_o)^{1/2}$	$\eta^* = \eta/(3\pi\eta_m) = \eta^\dagger/C^\dagger$
pressure (tensor)	$\underline{\underline{p}}^\dagger = \underline{\underline{p}}\sigma^3/E_o$	$\underline{\underline{p}}^* = \underline{\underline{p}}\sigma/mC^2 = \underline{\underline{p}}^\dagger/C^{\dagger 2}$
diffusivity	$\underline{\underline{D}}^\dagger = \underline{\underline{D}}(m/\sigma^2E_o)^{1/2}$	$\underline{\underline{D}}^* = \underline{\underline{D}}/C\sigma^2 = \underline{\underline{D}}^\dagger/C^\dagger$
kinetic energy	$E^\dagger = E/E_o$	$E^* = E/(m\sigma^2C^2) = E^\dagger/C^{\dagger 2}$
friction constant	$C^\dagger = C(m\sigma^2/E_o)^{1/2}$	C

For a system at any given packing density y $(= N\pi\sigma^3/6V)$ it is evident that the reduced properties in either scheme are functions only of the reduced shear rate. Thus we have the reduced flow curves

$$\eta^\dagger = f_{unction}(\dot{\gamma}^\dagger, y)$$

and

$$\eta^* = f_{unction}(\dot{\gamma}^*, y)$$

respectively. To scale from thermalised to hydrodynamic flow curves or *vice-versa* we need to know either

$$C^\dagger(\dot{\gamma}^\dagger, y)$$

or

$$E^*(\dot{\gamma}^*, y)$$

respectively, i.e. the mean friction coefficient in the damped Newtonian computation, or the mean kinetic energy in the Stokesian computation. This information is not presently available as only a very limited part of the isokinetic flow curve has so far been reported.

Our ultimate objective is to predict and hence understand the rheology of a well-characterised suspension of particles in Newtonian media under shear flow. In the hydrodynamic approximation the total stress in a sheared suspension can be resolved into two components. The first component, referred to as the intrinsic stress is, in the one-body approximation, the integral of the hydrodynamic stress around the surface of a flowing sphere with stick conditions. The second component is the osmotic stress from the interparticle interactions; if this is given from the scaling laws we can express the relative viscosity of a suspension as

$$\frac{\eta}{\eta_m} = 1 + \frac{5}{2}y + 3\pi\eta^* \tag{13}$$

Both the intrinsic and the osmotic contributions are generally shear-dependent, but having neglected many-body hydrodynamics, only the osmotic term is shear-rate dependent and this can be estimated from the scaling by the kinetic energy substitution.

$$\frac{\eta}{\eta_m} = 1 + \frac{5}{2}y + \eta^\dagger \left(\frac{E^*}{E^\dagger}\right)^{1/2} \tag{14}$$

Where E^* is the kinetic energy corresponding to the viscosity $\eta^\dagger$ at the shear rate $\dot{\gamma}^\dagger$ in units of $m\sigma^2 C^2$. The dependence of $C^\dagger$ on $\dot{\gamma}^\dagger$ at low $\dot{\gamma}^\dagger$-shear can be estimated from a consideration of the damped Newtonian hard-sphere NEMD computations.

$$\frac{3\bar{v}^2}{V}\frac{< 1 - f^2 >}{\bar{\Delta t}} = \eta^\dagger \dot{\gamma}^2 \tag{15}$$

(i.e. rate of heat out = rate of heat in)
where $(\Delta t)^{-1}$ is the mean scaling frequency and

$$C^\dagger_{eff} = \frac{< 1 - f >}{\bar{\Delta t}}$$

is the effective friction constant in HS units. Then

$$\bar{\Delta t} C^{\dagger 2}_{eff} + 2C^\dagger_{eff} - 2\eta^\dagger \dot{\gamma}^{\dagger 2}/3\rho = 0 \tag{16}$$

where $\rho = 6y/\pi$. When $C^{\dagger}_{eff} =< 1 - f^2 > /\bar{\Delta}t$, as used in the Berendsen rescaling scheme [6], then the quadratic term vanishes.

Equation (16) indicates that at low $\dot{\gamma}^{\dagger}$-shear the effective friction constant is decreasing at a rate proportional to $\dot{\gamma}^2$ when $\eta^{\dagger}$ and y are constants and $\bar{\Delta}t$ is small. At higher shear rates $C^{\dagger}_{eff}$ remains to be determined since the effect of $\bar{\Delta}t$ and large anisotropies is uncertain. The consequence of this initial observation is that the high shear-rate region of the isokinetic flow curve $\eta^{\dagger}(\dot{\gamma}^{\dagger})$ corresponds to the low shear-rate region of the mean-field Stokesian flow curve $\eta^{*}(\dot{\gamma}^{*})$ and *vice versa*.

The low-shear region of the isokinetic (or thermalised) flow curve at higher density is characterised by a disorder-order phase transition which stems from a perturbation of the equilibrium freezing transition for monodisperse spheres [7]. If the inversion suggested by the present analysis survives perturbations such as the inclusion of many-body hydrodynamics, interfacial non-Newtonian media effects and high-shear large normal differences, etc. then the steady-state quasi-equilibrium ordering transition corresponds to the well-known "shear-thickening" disordering transition seen in dense suspensions at higher shear rates [8].

The mean-field Stokesian scaling makes some interesting predictions that relate to experiment.

1. The reduced characteristic shear flow rate $\dot{\gamma}^{*} = \dot{\gamma}/C$ is the dimensionless group

$$\frac{4\sigma^2\dot{\gamma}\rho}{9\eta_m}$$

(where $\rho = Nm/V$). Apart from the constant this has long been recognised as the determining scaling group for characteristic shear rates such as the critical shear rate for the onset of the "shear-thickening" transition (Figure 1, data from H.A. Barnes [8]).

2. At low $\dot{\gamma}^{\dagger}$-shear the osmotic pressure goes to its equilibrium value approaching Newtonian behavior, but $C^{\dagger}$ is decreasing as $\dot{\gamma}^{\dagger 2}$. Thus, the osmotic pressure experienced at high shear in the $\dot{\gamma}^{*}$-flow curve is increasing as $C^{\dagger}$, resulting in ever increasing dilatancy in the high-shear region. The osmotic pressure $\underline{\underline{p}}^{*}$ is given by

$$\underline{\underline{p}}^{*} = \underline{\underline{p}}^{\dagger} E^{*}/E^{\dagger}$$

Figure 2, for example, shows what happens to the coexistence pressure of the ordering transition.

3. In the$\dot{\gamma}^{\dagger}$-flow curve both the kinetic energies and the diffusivities exhibit large normal differences with increasing shear-rate; this leads to low perpendicular diffusivities, long relaxation times and glassy-like non-steady behaviour. These time

dependent kinetic effects should be manifested at low shear in the $\dot{\gamma}^*$-flow curve. Thus, the low-shear ($\dot{\gamma}^*$) region is increasingly sensitive to Brownian motion constributions to E^* and $\underline{D}^*$, which are related by

$$\underline{D}^* = \underline{D}^\dagger \left(\frac{E^*}{E^\dagger} \right)^{1/2}$$

Figure 3, for example, shows what happens to the normal and parallel diffusivities.

Brownian motion effects have not been included in the calculations yet but can be accomodated into the scaling by effectively providing a baseline for E^*, $\underline{\underline{p}}^*$ and $\underline{D}^*$ which ensures a limiting Newtonian behaviour at low $\dot{\gamma}^*$ (given time!) and the connection to thermodynamic equilibrium as $\dot{\gamma}^* \to 0$.

References

[1] H.A. Barnes, M.F. Edwards and L.V. Woodcock, Chemical Engineering Science **42** 591 (1987).

[2] D.J. Evans and G.P. Morriss, Computer Physics Reports 1 297 (1984).

[3] L.V. Woodcock, CCP5 Information Quarterly **24** 29 (March 1987).

[4] J.J. Erpenbeck, Physica **118A** 114 (1983).

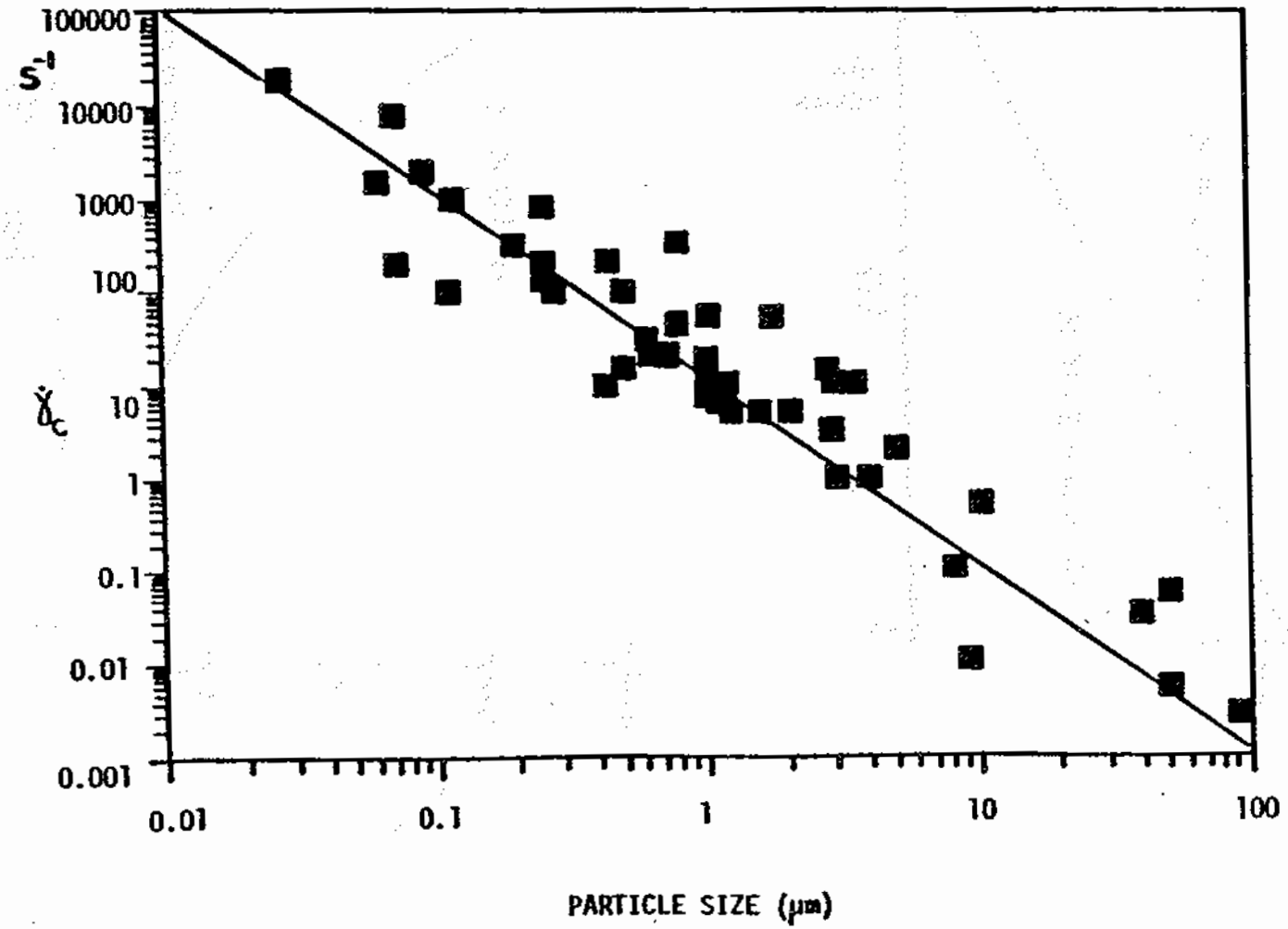
[5] Naitoh and Ono, J. Chem. Phys. **70** 4515 (1979).

[6] H.J.C. Berendsen, J.P.M. Postma, W.F. van Gunsteren, A. di Nola and J.R. Haak, J. Chem. Phys. **81** 3684 (1984).

[7] L.V. Woodcock, Phys. Rev. Lett. **54** 1513 (1985).

[8] H.A. Barnes, "Shear-thickening ('dilatancy') in suspensions of non-aggregating solid particles dispersed in Newtonian liquids: a review and practical guide", in press.

Figure 1.: A collection of literature data for the variation of the critical characteristic shear rate for the onset of shear-thickening (disordering) transition; the straight line corresponds to the γ^* scaling law.



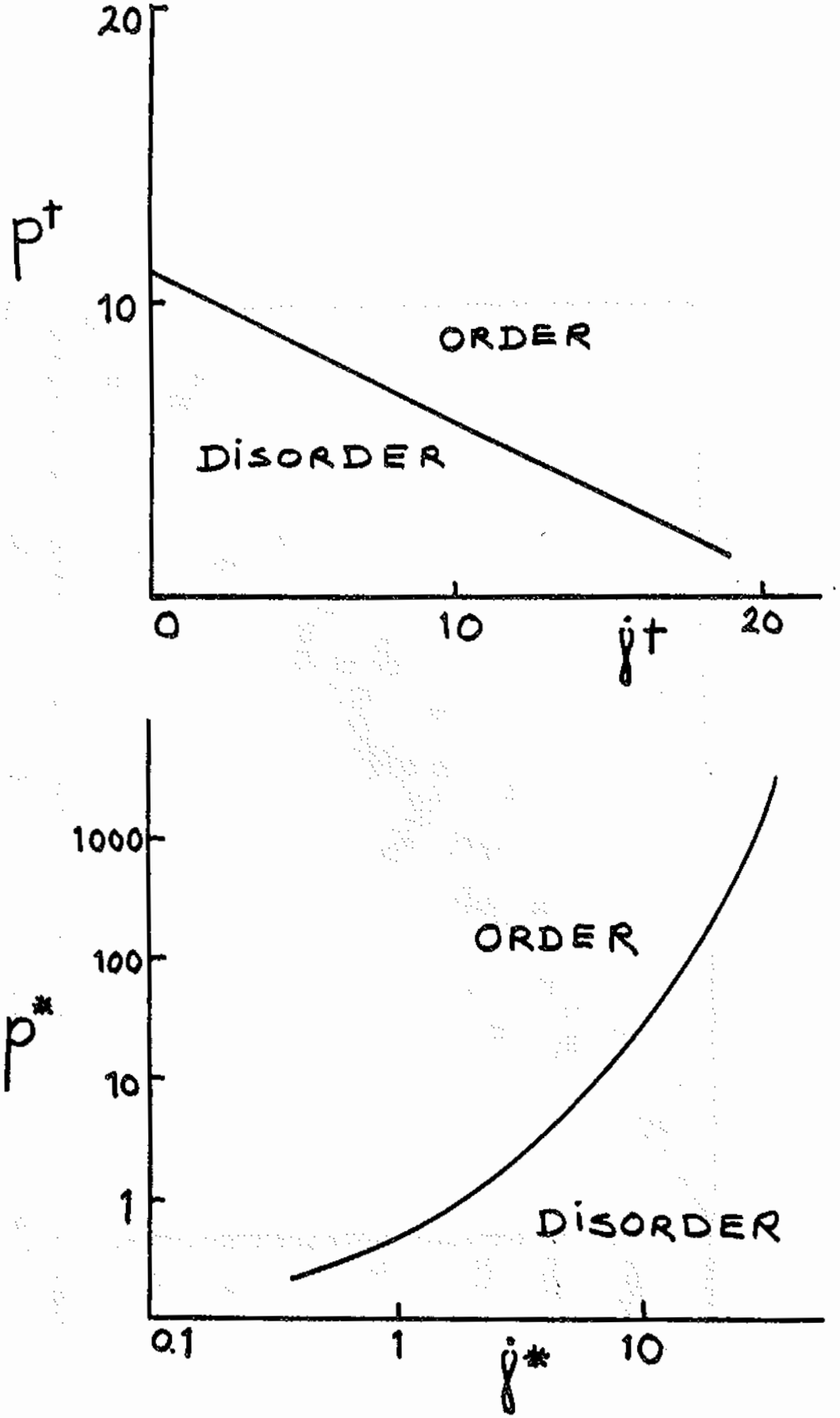


Figure 2: A prediction of the behaviour of the order-disorder transition for the hard-sphere model in isokinetic $p\dagger(\gamma\dagger)$ and mean field Stokesian form p^* (γ^*); the p^* axis reflects the large increase in dilatancy in this region.

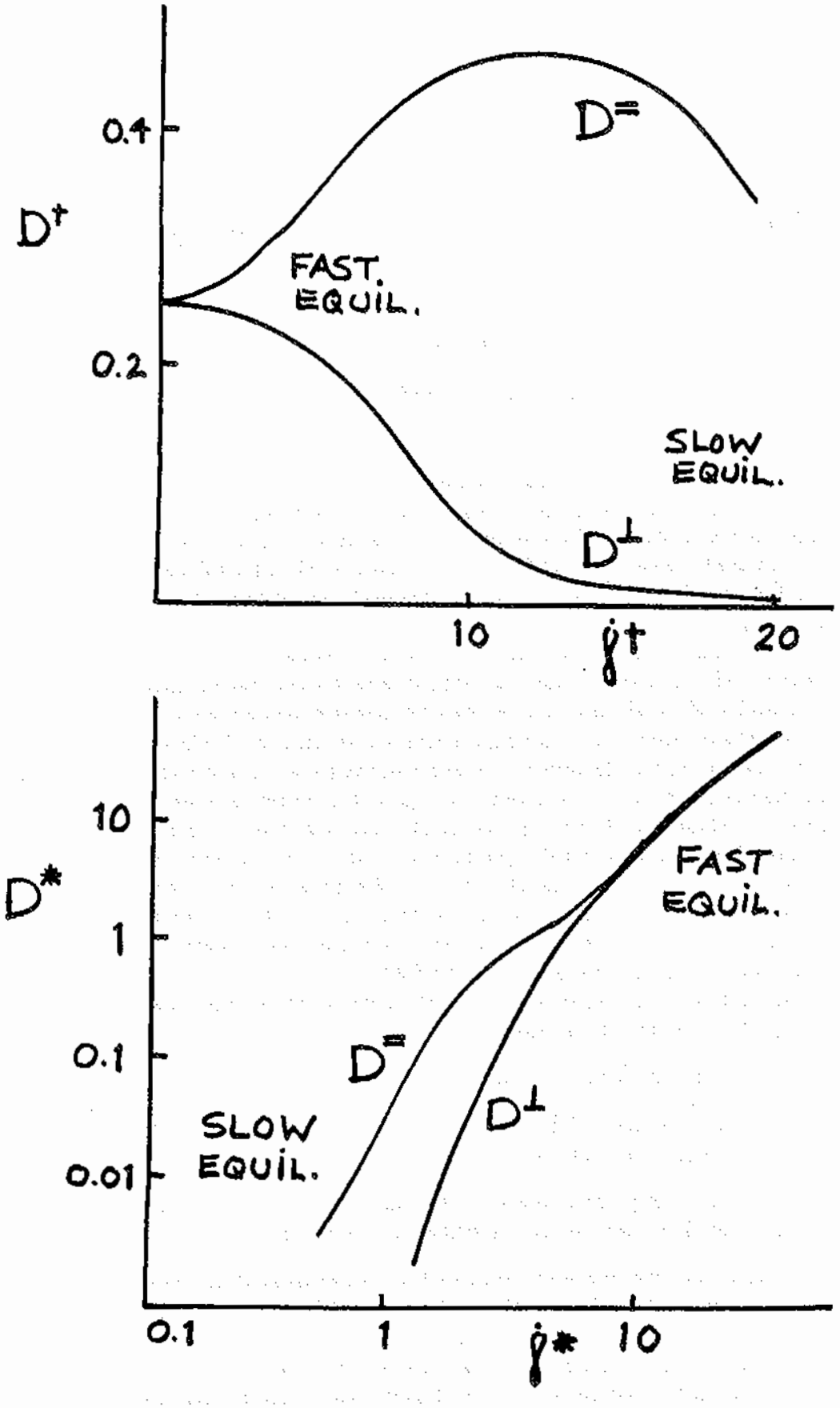


Figure 3: Behaviour of the hard-sphere diffusivity (roughly predicted from unpublished soft-sphere data) in both the isokinetic $D^{\dagger}(\gamma\dagger)$ and mean-field Stokesian D^{*} (γ^{*}) forms; $D^{\perp}$ and $D^{=}$ denote diffusivities perpendicular and parallel to the flow field respectively.

The Curious Case of the Bell that Rang Twice!

W. Smith

December 21, 1987

In this article we wish to report a curious observation regarding the properties of gaussian wavepackets, which has some bearing on their use in semiclassical molecular dynamics. The observation was made in the course of a study initiated at a CECAM meeting in 1986 and involved W. Smith, Konrad Singer and R. Kosloff. In this study we attempted to answer an apparently simple question: "How well do gaussian wavepackets describe thermal collisions between monatomic molecules?"

Our model was a very simple one-dimensional system. We imagined two molecules of equal mass, represented by gaussian wavepackets, initially at some distance apart, moving towards one another at a "thermal" velocity. The nature of the interaction between the particles was a short range repulsion, of the same effective range as the Lennard-Jones repulsive r^{-12} term, which we approximated using a gaussian function. The equations of motion of this system were obtained by applying the method described by E.J. Heller[1]. We intended to study this system for a range of particle masses, collision energies etc. and, by comparison with an exact solution, reveal the circumstances under which the gaussian wavepacket approximation broke down.

If this system were treated as entirely classical, we would expect the following effects as time progresses:

- the total energy of the system remains constant,
- the potential energy of the system rises to a maximum at the classical turning point and then diminishes,
- the kinetic energy of the particles decreases to zero at the turning point and rises again as the particles repel each other.

So what happens in the case of the collision between gaussians? In Figure 1 we have plotted the total, potential and (one-particle) kinetic energies for a system approximating to two hydrogen atoms. [1]

The first point to mention is the constant value of the total energy; this much at least is satisfactory. The kinetic and potential energies are, to say the least, surprising. The potential energy curve shows two

[1] These figures use atomic units throughout; the time axis numbers the MD timesteps, which were 0.1 atomic units.

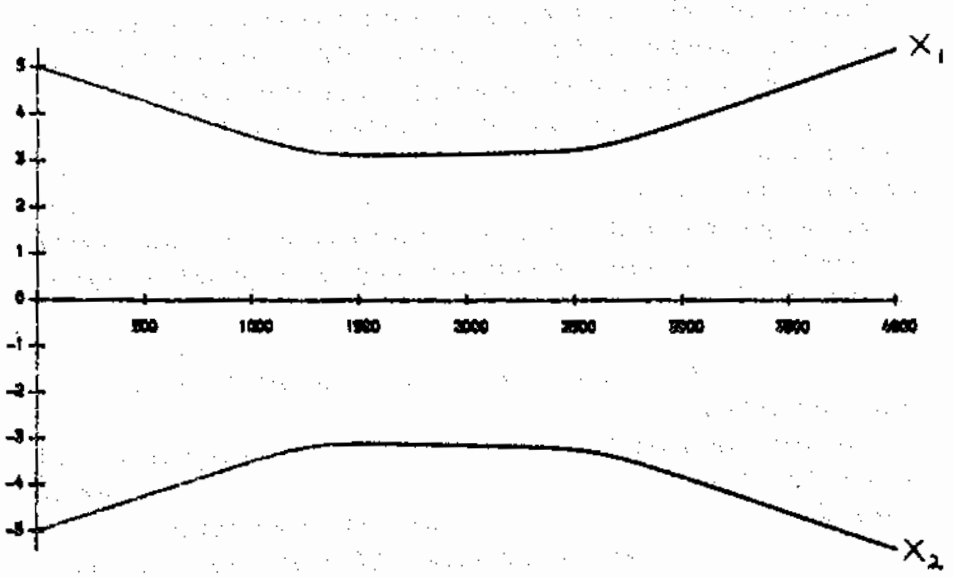


FIGURE 3. Position vs. Time.

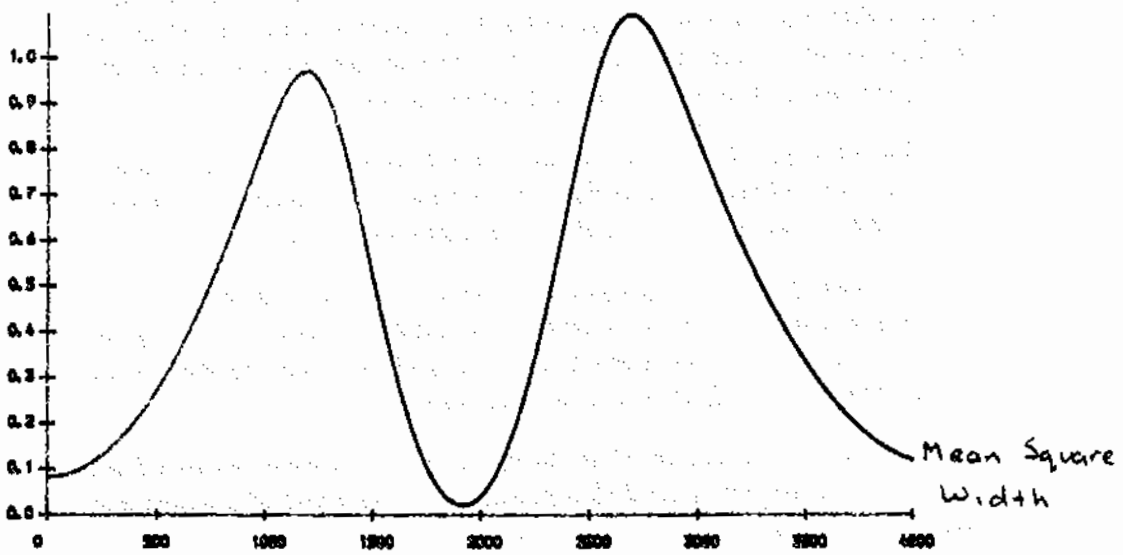


FIGURE 4. Mean Square Width vs. Time

reversed. Thereafter the wavepacket contracts dramatically to a minimum. (It would appear that this minimum occurs at the same point in time as one which is obtained from the intersection of the asymptotes of the trajectory given in Figure 3. This might therefore be an unambiguous definition of the turning point for this system.)

The point of minimum contraction of the wavepacket coincides with the reduction of the potential energy of the system and also with an increase in the total kinetic energy. We note however that the momenta of the wavepackets at this point are still small and we must conclude that the bulk of the kinetic energy is contained within the wavepackets themselves, and is expressed as a function of the parameter A. Thus we see that, to a large extent, the width parameter temporarily absorbs the energy of the collision. In this regard it acts suspiciously like an internal degree of freedom.

Once past this point, the wavepacket expands again rapidly, so quickly in fact that it appears to outrun the speed of recession of the two wavepackets. A second collision is inevitable, with many of the features of the first, but with an important difference. The first collision **decelerates** the linear motion of the wavepackets, while the second **accelerates** the motion in the opposite direction. Evidence in favour of this explanation comes from Figure 2, in which there is an initial rapid decrease in momentum for both wavepackets followed by a very slow "recoil" period (when the wavepackets are extremely narrow) and finally there is a sudden acceleration of the wavepackets as the expansion of the wavepackets causes them to interact strongly and thus repel each other.

In conclusion we see that gaussian wavepacket dynamics introduces a new feature into the dynamics of molecular systems, which fortunately are explicable but raise questions as to their suitability for semiclassical MD. The feature is the dynamical variability of the wavepacket width, which gives rise to the following novel phenomena:

- The capacity of the wavepacket to act as an additional "energy store" in collisions, absorbing the energy of collisions and also affecting to the momentum of the recoiling particles.

- The capacity of the wavepacket to expand rapidly and thus causing the particles to interact sooner than expected and conversely, to contract when the particles do interact and thus prolong the period of time over which they can be said to interact.

Since there can be little doubt that the dynamics of gaussian wavepackets would be essentially classical if this feature were removed (the equations of motion show this) one wonders what modifications must be made to remove these strange phenomena from the model and yet retain some semblance of quantum behaviour; for it must be said that these phenomena are **not** present in exact simulations using discrete Fourier transform methods[3].

References

[1] E.J. Heller, J. Chem. Phys. **62** (1975) 1544 and **64** (1976) 63.

[2] K. Singer and W. Smith, Molec. Phys. **57** (1986) 761.

[3] R. Kosloff, private communication.

C-language MD code running on PC ??
How forces() procedure looks like ?

Jacek Mościński, Witold Dzwinel and Jacek Kitowski
Institute of Computer Science AGH
30-059 Krakow, Poland

The current MD simulation activity is FORTRAN oriented and emphasized on vector and parallel processing and on large supercomputers. So do the papers in this World renowned Newsletter. In this note we would like to go a little against the trend and to show that a modest but reasonable (we believe !) simulational work can be done in other than FORTRAN languages and - what is more important for computational scientists in some geographic regions - in a microcomputers world (which is evolving fast toward several MFLOPs on your desk).

With a wide availability of personal computers and workstations C language becomes more and more popular. C is also the main language in large scientific projects in parallel processing like the Caltech/ JPL Concurrent Computation Project. We have already devised a conventional constant energy MD C-language program suitable for PC/workstation environment. The code, which we call MD3DLJ, is intended for simulation of monoatomic molecule mixtures. It uses 12/6 L-J site-site shifted potential functions and a Verlet leapfrog algorithm for centre-of-mass motion. Program calculates system average configuration energy, kinetic energy and virial, pressure, temperature, and associated R.M.S. deviations, MSDs and RDFs.

In this note we present the forces() procedure and comparison of the MD3DLJ run timings for different numbers of particles, that we believe may be of interest to readers of the CCP5 Newsletter .

The procedure forces() is written close to the OLYMPUS standard [1] (see Table 1) and communicates with its environment using external declarations (lines 9 - 53). Internal variables are defined in lines 62 - 68. According to the convention used, the external variables begin with big letters, while internal ones with small letters. We believe that the naming convention

reflects meaning of the variables.

In the procedure the link-cell method [2,3] is used for determination of neighboring particles of the current one. If we let *Link_head[icell]* be the head-of-chain table entry for chaining cell *icell* and let *Link_list[i]* be the particle linked to the *i* one then the procedure for sorting particles into lists for each chaining cells by means of address sorting is represented by lines 84 - 93. *No_parts* and *No_cells* mean the number of particles and the number of cells respectively.

Range of the computational box is [0, *No_cells_x*], [0, *No_cells_y*], [0, *No_cells_z*], where *No_cells_x*, *No_cells_y*, *No_cells_z* are the numbers of cells in x, y and z direction respectively. According to the range of the computational box the cell index determination for each particle is simple due to direct substitution of the particle coordinates *X[i]*, *Y[i]*, *Z[i]* into the cell coordinates *icell_x*, *icell_y* and *icell_z* (line 87).

The procedure forces() is composed of two main loops for the individual particle forces calculation. The primary loop is organized over all cells of the computational box (lines 100 - 293). Its purpose is also to determine the head of the link-list for the current cell (see line 103).

Calculations of the individual particle forces are done within the secondary loop. In the presented procedure, in order to increase overall efficiency, the secondary loop is divided into three kinds. The calculations with the secondary loop are bypassed however, while the current cell is empty (see line 104). Each kind of the secondary loop determines forces on the current particle derived from the well defined and separated sets of cells. Thus the first kind of the secondary loop (lines 109 - 144) calculates forces derived from the particles belonging to the same cell as the current particle. The second kind of the secondary loop takes into account interactions from the particles belonging to the cells with other z-coordinates than the current one (see lines 146 - 216). To avoid double calculations of interactions between the particles only half cells out of the neighboring the current cell is considered. As a consequence the only particles from cells lying above (in z-direction) the current one are examined. The third kind reflects interactions derived from the particles belonging to the cells lying on the same level as the current cell and also half of them is considered only.

Coordinates of the current cell are modified at the end of the primary loop (lines 284 - 292). Coordinates of the neighboring cells (called *jcell_x*, *jcell_y* and *jcell_z* respectively) are determined for every kind of the secondary loop separately using the arrays *Neigh_x*, *Neigh_y* and *Neigh_z* (see lines 153 - 155 and 226 - 227), defined outside the procedure and constant during the simulation. Index of the neighboring cell under investigations, *jcell*, is calculated in lines 164 and 235 - 237 in order to define the head of the link list (*Link_head[jcell]*). Instead of classical calculations, the index is defined with the arrays *Ncell_bound_x[jcell_x]*, *Ncell_bound_y[jcell_y]* and *Ncell_bound_z[jcell_z]*, predefined outside the procedure.

Periodic boundary conditions (minimum image convention) are applied in the classical way (see lines 173 - 175 and 244), they are organized with help of the arrays *Iconvert_x[jcell_x]*, *Iconvert_y[jcell_y]*, *Iconvert_z[jcell_z]* (lines 159 - 161 and 231 - 232).

In such an approach consisting of the three kinds of the secondary loop significant increase of the overall efficiency is achieved according to the careful choice of necessary instructions suitable for every kind of the secondary loop separately (compare lines 153 - 161 with 226 - 232 and with absence of similar instructions in the first kind of the secondary loop).

In conventional approaches the lenghts of sides of the cells of the chaining mesh are always greater than or equal to the cutoff radius. The latter case is the better one, because unnecessary calculations of the Euclidean distance are reduced.

To reduce the computer time while the side length of the cells is greater than the cutoff radius two techniques have been introduced. The first one consists of checking of coordinates of the neighboring particle. If differences in coordinates between the current particle and the neighboring one are less than the cutoff radius *Rcut* then the Euclidean distance is evaluated and compared with square of *Rcut* (i.e. *Rcut2*), otherwise no distance is calculated (see lines 184 - 190 and 250 - 254). We call this technique the "cube technique" since only particles taken for further analysis are those which differences in their coordinates and the current particle lie inside the cube of 2 * *Rcut* sides.

In the second technique tiny cells of the lenghts of sides less than the cutoff radius are introduced. Obviously, more cells in each direction have to be taken into account. If the tiny cells are generated by division each of the previously defined cells, the more close particles to the current one are examined only, because part of the tiny cells can be neglected. Assume for example that every side of 27 cells is divided into 2 parts, then 125 out of 216 cells is taken into considerations only. In that case the "tiny cell technique" consumes 0.58 of the time spent on calculations with classically organized cells. In practice (see Table 3) advantage of the method is not so high according to more computational time spent for cells searching. The algorithm is applied by the proper definition of coordinates of the neighboring cells (i.e. *Neigh_x*[*jc*], *Neigh_y*[*jc*] and *Neigh_z*[*jc*] - used in lines 153 - 155, 226 - 227) and by secondary loop indices (lines 150, 224).

In the procedure kinds of molecules (or species, of number *No_specs*) are represented by array *Kind_specs*[*i*] of dimension equal to *No_parts* (lines 117, 176 and 245). Due to the array of the molecule kinds the simulation time does not depend in principle on the number of species, in contrast with e.g.[4].

The Lennard - Jones pair potentials are represented by values stored in *No_specs* * (*No_specs* + 1) / 2 number of arrays which is equivalent to the number of potentials caused by different types of molecules in the mixture. The values are available by the array of pointers *pPot*[*kind*], where *kind* means the kind of pair potential. The kinds of pair potential are represented by the array *Kind_pot* (see lines 126, 193 and 257). The number of elements in the arrays of potential values is a parameter of the program (set to 2000 at present). The distance interval covered by the grid starts from square of potential "hard-core" (*Rhard2*[*kind*]) which assumes to be a fraction (0.5 at present) of σ(*kind*) L-J parameter, where *kind* means the kind of potential. The end of the distance interval is equal to square of the cutoff radius (*Rcut2*). Due to the different kinds of molecules, the cutoff radius (*Rcut*) is assumed to be multiple (2.5 at present) of the greatest σ(*kind*).

Elements from the potential tables are chosen according to the kind of potential (kind of interacted molecules) and to the distance between the molecules. Computational savings are made by tabulating the values of potentials at uniform intervals Δr^2

avoiding computations of square roots during elements getting from tables. Thus in the *kind*-th table with the pointer *pPot*[*kind*], values of the expression $2 * U/\Delta r^2$ are tabulated, where U is the L-J potential function of distance between the particles. In the procedure $1/\Delta r^2$ is called *Step2_pot_rev*.

To determine $2 * dU/dr^2$ and consequently force on the *i*-th particle (*Force_x*[*i*], *Force_y*[*i*] and *Force_z*[*i*]), values of two neighboring grid points are chosen from the tables. By the way the potential energy (*Energ_pot*) and virial (*Virial*) are calculated as well.

Computations of forces on the particles with the method described above are applied into the procedure in lines 126 - 138, 192 - 206 and 256 - 269.

Tables 2 and 3 present some timings for runs of the program MD3DLJ. They were obtained on a DSI-020 plug-in board for microcomputers compatible with IBM PC/AT or XT. The board consists of Motorola's 68020 CPU, 68881 hardware floating point coprocessor, 1MB of RAM and clock 12.5 MHz. Another hardware applied was an IBM PC/AT-turbo (10MHz) clone (supported with 80287 and MS DOS 3.10 operating system). The following compiler were applied: SVS C-compiler (version 2.6) for DSI-020 board and Microsoft C-compiler (version 3.0) for the IBM PC/AT clone. The simulated mixture consists of equal number of Ar-Kr molecules in temperature 116 K with molar volume $34.33\ 10^{-6}\ m^3$. The timestep used is equal to 10 fs. The achieved R.M.S. fluctuation in total energy of the system of 2916 particles averaged over steps 3000-5000 (while the temperature scaling stops at 2000 step) are less than 0.01%.

Table 2 shows that linear dependence of computer time for a single MD timestep on the number of particles is conserved well. Table 3 presents average values of computer time for a single timestep per particle. They are averaged over runs with different numbers of particles (presented in Table 1, except 256). Version 1 of the procedure forces() is presented in Table 1. Version 2 is written in C language on a base of [3] with mixture simulation and potential tables extensions described above. Version 3 is similar to Version 1 however without the "cube technique" (i.e. conditions in instructions 184 and 250 were reduced to square distance checking only). For indication of "tiny cell technique" applications a side cell ratio (SCR) parameter is used. It

reflects the side ratio of the classical cells to the tiny cells.

Comparison of Version 3 with Version 2 shows that 0.2 computational time savings are observed with the secondary loop division into three kinds (parts, for SCR = 1). Introducing the "tiny cell technique" the savings increase to 0.35 achieving maximum for SCR = 2. The "cube technique" seems to be more profitable (compare Version 1 with Version 3 for SCR = 1) and 0.45 savings are obtained. Application of the "tiny cell technique" together with the "cube" one is not useful since savings of computational time decrease monotonically with SCR increase in that case. Another feature of the "cube technique" is weak dependence (in comparison with Version 2 or 3) of computational time for a single timestep per particle on proper determination of cutoff radius with reference to the cell side.

We are grateful to Prof. Jan Klamut of the Polish Academy of Science for arranging a partial financial support for our activity under the project number CPBP 01.12-25/P/87.

References.

[1] J.P. Christiansen and K.V. Roberts, "A Standard Control Utility Package for Initial Value FORTRAN Programs" Comput. Phys.Commun. 7(1974)245-270.
[2] R.W. Hockney and J.W. Eastwood, "Computer Simulation Using Particles" McGraw Hill (1981) p.277.
[3] W.Smith, "Fortran Code for the LINK-CELL Method" CCP5 Information Quarterly for Computer Simulation of Condensed Phases No.20(1986)52 Daresbury Laboratory.
[4] W. Smith, "The Program ADMIXT, A Molecular Dynamics Program for the Simulation of Monoatomic Liquid Mixtures" CCP5 Program Library June 1983.

Table 1. Listing of the procedure forces()

```
1                                                                    /*
2   DECLARATION OF EXTERNAL VARIABLES                        file gmd20a.txt
3   ═══════════════════════════════
4
5   CONSTANT DEFINITIONS
6   ────────────────────                                             */
7   #define    MAX_SPECS     4
8                                                                    /*
9   LIST OF EXTERNAL VARIABLES
10  ──────────────────────────                                       */
11                                                                   /*
12  2. PROPERTIES OF THE SYSTEM
13
14  2.1. PRINCIPAL PHYSICAL PARAMETERS                               */
15
16  extern     double    *X, *Y, *Z;
17  extern     int       No_parts, No_specs;
18  extern     double    *pPot[];
19                                                                   /*
20  2.2. SECONDARY PHYSICAL PARAMETERS                               */
21
22  extern     double    *Force_x, *Force_y, *Force_z;
23  extern     double    Energ_pot;
24  extern     double    Virial;
25                                                                   /*
26  3. NUMERICAL VARIABLES
27
28  3.1. NUMERICAL PARAMETERS
29                                                                   */
30  extern     int       Kind_pot[][MAX_SPECS];
31  extern     int       *Kind_specs;
32  extern     double    Step2_pot_rev;
33                                                                   /*
34  3.2. LINKED LIST                                                 */
35
36  extern     int       *Link_head, *Link_list;
37  extern     int       *Neigh_x, *Neigh_y, *Neigh_z;
38                                                                   /*
39  3.3. CUTOFF RADIUS VARIABLES AND CELL MESH PARAMETERS
40                                                                   */
41  extern     double    Rcut, Rcut2;
42  extern     double    Rhard2[];
43  extern     int       No_cells_x, No_cells_y, No_cells_z, No_cells;
44  extern     int       No_neigh_cells1;
45  extern     int       No_neigh_up;
46                                                                   /*
47  3.5. OTHER NUMERICAL PARATEMETRS                                 */
48
49  extern     int       *Ncell_bound_x, *Ncell_bound_y, *Ncell_bound_z;
```

```
50  extern     int       *Iconvert_x, *Iconvert_y, *Iconvert_z;
51
52                                                                    /*
53  END OF EXTERNAL DECLARATIONS
54  ============================                                      */
55
56
57  forces ()                                                /* module c2s2 */
58                                                                    /*
59  2.2 CALCULATE LENNARD - JONES FORCES                              */
60
61  {
62  register int  i, j, kind, i_kind, idist, jc, jcell, icell;
63  int           icell_x, icell_y, icell_z, jcell_x, jcell_y, jcell_z;
64  int           ifirst, k;
65  int           iconv_x, iconv_y, iconv_z;
66  double        difpot, f_x, f_y, f_z, xi, yi, zi;
67  double        dif_x, dif_y, dif_z, dist2, *poin;
68  double        r_xa, r_ya, r_za, r_xs, r_ys;
69                                                                    /*
70  ------------------------------------ FORCES - BEGIN ------------------*/
71
72                                                                    /*
73  1.INITIALIZE POTENTIAL ENERGY,VIRIAL AND FORCE ARRAYS
74                                                                    */
75      Energ_pot = 0;
76      Virial    = 0;
77      for    (i = 0; i < No_parts; i++)
78             {
79             Force_x[i] = 0; Force_y[i] = 0; Force_z[i] = 0;
80             }
81                                                                    /*
82  2.CALCULATE LINK CELL INDICES
83                                                                    */
84      for    (i = 0; i < No_cells; i++) Link_head[i] = -1;
85      for    (i = 0; i < No_parts; i++)
86             {
87             icell_x = X[i];   icell_y = Y[i];    icell_z = Z[i];
88             icell   = icell_x + No_cells_x
89                       * (icell_y + No_cells_y * icell_z);
90             j       = Link_head[icell];
91             Link_head[icell] = i;
92             Link_list[i]     = j;
93             }
94                                                                    /*
95  3.CALCULATE THE INDIVIDUAL PARTICLE FORCES
96
97  3.1 PRIMARY LOOP OVER ALL CELLS
98                                                                    */
99      icell_x = 0; icell_y = 0; icell_z = 0;
```

```
100       for (icell = 0; icell < No_cells; icell++)  /* Start of 3.1 primary
101                                                                   loop  */
102             {
103             ifirst = Link_head[icell];
104             if(ifirst >= 0)           /* Bypass current cell if empty */
105                  {
106                                                                     /*
107   3.2 SECONDARY LOOPS OVER CELLS
108
109   3.2.1 SECONDARY LOOP OVER CURRENT CELL
110                                                                     */
111                  i = ifirst;
112
113                  do                          /* Start of 3.2.1 do loop */
114                       {
115                       j     = Link_list[i];
116                       xi    = X[i];   yi  = Y[i];    zi  = Z[i];
117                       i_kind = Kind_specs[i];
118
119                       while (j != -1)     /* Start of 3.2.1 while loop */
120                            {
121                            dif_z=Z[j]-zi; dif_x=X[j]-xi; dif_y=Y[j]-yi;
122                            if ((dist2 = dif_x * dif_x
123                                      + dif_y * dif_y + dif_z * dif_z)
124                                                          < Rcut2  )
125                                 {
126                                 kind = Kind_pot[i_kind] [Kind_specs[j]];
127                                 idist= (dist2 - Rhard2[kind])
128                                              * Step2_pot_rev;
129                                 poin = pPot[kind] + idist;
130                                 difpot = *(poin + 1) - *poin;
131                                 f_x    = difpot * dif_x;
132                                 f_y    = difpot * dif_y;
133                                 f_z    = difpot * dif_z;
134                                 Energ_pot += *(poin + 1) + *poin;
135                                 Virial    += dist2 * difpot;
136                                 Force_x[i]-= f_x;  Force_x[j]+= f_x;
137                                 Force_y[i]-= f_y;  Force_y[j]+= f_y;
138                                 Force_z[i]-= f_z;  Force_z[j]+= f_z;
139                                 }
140                            j = Link_list[j];
141                            }              /*  End of 3.2.1 while loop */
142                       i = Link_list[i];
143                       }
144                  while (i != -1);            /*  End of 3.2.1 do loop */
145                                                                     /*
146   3.2.2 SECONDARY LOOP OVER NEIGHBORING CELLS LYING ABOVE
147         THE CURRENT ONE  (IN Z DIRECTION)                           */
148
149
```

```
150                 for (jc = 0; jc < No_neigh_up; jc++)   /* Start of
151                                                secondary 3.2.2 for loop  */
152                     {
153                     jcell_x = icell_x + Neigh_x[jc];
154                     jcell_y = icell_y + Neigh_y[jc];
155                     jcell_z = icell_z + Neigh_z[jc];
156                     i       = ifirst;
157
158                                     /*   Minimum image convention */
159                     iconv_x  = Iconvert_x[jcell_x];
160                     iconv_y  = Iconvert_y[jcell_y];
161                     iconv_z  = Iconvert_z[jcell_z];
162
163                                     /*  Index of neighboring cell */
164                     jcell    = Ncell_bound_x[jcell_x]
165                              + Ncell_bound_y[jcell_y]
166                              + Ncell_bound_z[jcell_z];
167                     k        = Link_head[jcell];
168
169                     if (k == -1)  continue;     /* Bypass empty cell */
170
171                     do              /* Start of 3.2.2 outer do loop */
172                         {
173                         xi     = X[i]-iconv_x;
174                         yi     = Y[i]-iconv_y;
175                         zi     = Z[i]-iconv_z;
176                         i_kind = Kind_specs[i];    j = k;
177                         r_xa   = Rcut + xi;
178                         r_ya   = Rcut + yi;
179                         r_za   = Rcut + zi;
180                         r_xs   = xi - Rcut;  r_ys = yi - Rcut;
181
182                         do          /* Start of 3.2.2 inner do loop */
183                             {
184                             if ( Z[j] < r_za &&
185                                 X[j] > r_xs && Y[j] < r_ya &&
186                                 Y[j] > r_ys && X[j] < r_xa &&
187                                 (dist2 = (dif_x = X[j] - xi) * dif_x
188                                  + (dif_y = Y[j] - yi) * dif_y
189                                  + (dif_z = Z[j] - zi) * dif_z)
190                                                        < Rcut2 )
191                                 |
192                                 kind   =
193                                 Kind_pot[i_kind] [Kind_specs[j]];
194                                 idist  =
195                                 (dist2 - Rhard2[kind])
196                                       * Step2_pot_rev;
197                                 poin   = pPot[kind] + idist;
198                                 difpot = *(poin + 1) - *poin;
199                                 f_x    = difpot * dif_x;
```

```
200                                     f_y    = difpot * dif_y;
201                                     f_z    = difpot * dif_z;
202                                     Energ_pot += *(poin + 1) + *poin;
203                                     Virial    += dist2 * difpot;
204                                     Force_x[i]-= f_x; Force_x[j]+= f_x;
205                                     Force_y[i]-= f_y; Force_y[j]+= f_y;
206                                     Force_z[i]-= f_z; Force_z[j]+= f_z;
207                                     }
208                                 j = Link_list[j];
209                                 }
210                             while (j != -1);     /* End of 3.2.2 inner do
211                                                                 loop */
212                             i = Link_list[i];
213                             }
214                         while (i != -1);   /* End of 3.2.2 outer do loop */
215
216                         }                  /* End of secondary 3.2.2 for loop */
217
218
219                                                                         /*
220     3.2.3 SECONDARY LOOP OVER NEIGHBORING CELLS LYING ON THE SAME
221           LEVEL  AS THE CURRENT ONE                                     */
222
223                                     /* Start of secondary 3.2.3 for loop */
224                 for  (jc = No_neigh_up; jc < No_neigh_cells1; jc++)
225                     {
226                     jcell_x = icell_x + Neigh_x[jc];
227                     jcell_y = icell_y + Neigh_y[jc];
228                     i       = ifirst;
229
230                                         /* Minimum image convention */
231                     iconv_x    =  Iconvert_x[jcell_x];
232                     iconv_y    =  Iconvert_y[jcell_y];
233
234                                        /* Index of neighboring cell */
235                     jcell = Ncell_bound_x[jcell_x]
236                           + Ncell_bound_y[jcell_y]
237                           + Ncell_bound_z[icell_z];
238                     k     = Link_head[jcell];
239
240                     if (k == -1)  continue;     /* Bypass empty cell */
241
242                     do                 /* Start of 3.2.3 outer do loop */
243                         {
244                         xi = X[i]-iconv_x; yi= Y[i]-iconv_y; zi= Z[i];
245                         i_kind = Kind_specs[i]; j = k;
246                         r_xs = xi - Rcut; r_ya = Rcut + yi;
247
248                         do             /* Start of 3.2.3 inner do loop */
249                             {
```

```
250                             if ( X[j] > r_xs && Y[j] < r_ya &&
251                             (dist2 = (dif_x = X[j] - xi) * dif_x +
252                                     (dif_y = Y[j] - yi) * dif_y +
253                                     (dif_z = Z[j] - zi) * dif_z )
254                                                         < Rcut2 )
255                                 {
256                                 kind   =
257                                 Kind_pot[i_kind] [Kind_specs[j]];
258                                 idist  =
259                                 (dist2-Rhard2[kind])* Step2_pot_rev;
260                                 poin   = pPot[kind] + idist;
261                                 difpot = *(poin + 1) - *poin;
262                                 f_x    = difpot * dif_x;
263                                 f_y    = difpot * dif_y;
264                                 f_z    = difpot * dif_z;
265                                 Energ_pot += *(poin + 1) + *poin;
266                                 Virial    += dist2 * difpot;
267                                 Force_x[i]-= f_x;  Force_x[j]+= f_x;
268                                 Force_y[i]-= f_y;  Force_y[j]+= f_y;
269                                 Force_z[i]-= f_z;  Force_z[j]+= f_z;
270                                 }
271                             j = Link_list[j];
272                             }
273                         while (j != -1);  /* End of 3.2.3 inner do
274                                                                 loop */
275                         i = Link_list[i];
276                         }
277                     while (i != -1);   /* End of 3.2.3 outer do loop */
278                     }            /* End oof secondary 3.2.3 for loop */
279                 }                   /*  End of current cell bypass */
280
281                                                                 /*
282                                 Current cell index control section
283                                                                 */
284                 ++icell_x;
285                 if    ( icell_x >= No_cells_x)
286                       {
287                       icell_x = 0;  ++icell_y;
288                       if    (icell_y >= No_cells_y)
289                             {
290                             icell_y = 0;  ++icell_z;
291                             }
292                       }
293           }                         /*   End of primary 3.1 for loop */
294
295     Energ_pot /= 4. * Step2_pot_rev;
296
297                                                                 /*
298 ------------------------------------ FORCES - END ------------------*/
299 }
```

Table 2. Typical Timings for MD3DLJ Program. Version 1, SCR = 1.
(average values for first 100 steps of simulation)

No. of particls	No. of cells	Rcut in program units	MD timestep [s]		MD timestep per particle [s]	
			DSI-020	PC/AT	DSI-020	PC/AT
6912	8*8*8	0.99	145.0		0.021	
5324	7*7*7	0.95	114.7		0.022	
4000	6*6*6	0.89	89.4		0.022	
2916	6*6*6	0.99	61.2	203.4	0.021	0.070
2048	5*5*5	0.93	44.7	149.7	0.022	0.073
1372	4*4*4	0.85	31.7	107.8	0.023	0.079
864	4*4*4	0.99	18.2	60.3	0.021	0.070
500	3*3*3	0.89	11.2	37.7	0.022	0.075
256	2*2*2	0.74	6.7	23.1	0.026	0.090

Table 3. Comparison of computer time averages for MD timestep per particle in [s]

SCR	Procedure Versions					
	1		2		3	
	DSI-020	PC/AT	DSI-020	PC/AT	DSI-020	PC/AT
1	0.0218	0.0734	0.0403	0.1278	0.0321	0.1054
2	0.0284	0.0888			0.0265	0.0816
3	0.0388	0.1142			0.0315	0.0902

Version 1: "cube technique" mixed with "tiny cell technique"
Version 2: see explanation in the text
Version 3: "tiny cell technique" only

C-language code for irregular close packing of spheres. A procedure for nearest neighbour determination.

Jacek Mościński and Monika Bargieł

Institute of Computer Science, AGH
30-059 Kraków, Poland

Irregular close packing of hard spheres, also known as random close packing, has been extensively studied due to its importance as models for particulate systems in a wide variety of fields. The sphere packing problem is found in many domains of engineering, for example metallurgy, ceramics, soil science, physics, chemistry. Equal sized randomly packed spheres have also been suggested as models for liquid and glassy states [1,2].

We have just devised a C-language program suitable for PC/workstation environment, simulating a close, irregular packing of equal hard spheres. The algorithm applied is described in the literature [2,3] and has been already programmed in FORTRAN by Jodrey and Tory [4]. In contrast to the origin, our program is written according to specified methodology, close to the standard programming system OLYMPUS [5-7]. The standarization of the program structure, making it modular and as independent as possible of the computer system, speeds up construction of the program and facilitates its further modifications. Additionally the program is provided with restart and on-line data saving facilities as well as with the semigraphical lineprinter output and input data control possibility. The overall efficiency of the program is increased due to careful investigations resulting in the procedure presented below which appears to be the most time-consuming part of the program.

The irregular close packing of hard spheres is simulated in a cubic box with periodic boundary conditions. The simulation is started with a set of randomly distributed points taken as initial positions of the sphere centres. The algorithm consists of a number of subsequent iteration steps. In each step the two closest spheres in the ensemble are separated by moving the both spheres along the line joining their centres. Simultaneously the sphere diameters are modified according to current configuration of the system. The program determines final positions of the sphere centres and calculates the sphere diameter and corresponding packing density.

In this note we present the innermost procedure of the program, called *min_separation(ipart_p)*. External variables declared in lines 1-38 assure the procedure communication with its environment. According to the conventions

assumed names of all external variables begin with big letters contrary to the local variable names in which small letters are used only. The local variables of the procedure are defined in lines 48-56. The reader can also notice the application of FORTRAN-like first letter convention, concerning variable types. The decimal numbering of the program procedures, sections, subsections and groups of external variables arises from the conventions used.

The presented procedure is involved always just after the chosen sphere *ipart_p* has been moved. Its purpose is to find the nearest neighbour of the sphere (i.e. the sphere closest to *ipart_p*).

In the program two lists are applied to simplify the nearest neighbour searching and the worst overlap determination. The first one, called linked-list, sorts the spheres according to the coordinates of their centres. It is used to locate the spheres, centres of which lie within a specified cutoff radius. The second one, named sorted-list, keeps the spheres in increasing order of distances to their nearest neighbours. The first element of the sorted-list always indicates the pair of the closest spheres in the ensemble, which is to be taken for processing in the current iteration.

The nearest neighbour of the current sphere is searched from among a small subset of the total number of spheres only. For this purpose the link-cell method [8] is used together with the linked-list structure. According to this method the computational box is divided into $No\_cells\_1D^3$ elementary cells. Consequently *Link_head*[*icell*] is the head-of-chain table entry for chaining cell *icell*. *Link_list*[*i*] is the link coordinate for the sphere *i* and indicates another sphere lying in the same elementary cell as *i*. Additionally the *Link_inv* table is introduced. It keeps the spheres in reverse order relative to *Link_list* making possible the backward searching of the linked-list. Ranges of the computational box side lengths are [0,*No_cells_1D*] in each direction. That implies that the unit of length of the computational box is equal to the length of the elementary cell. Under this assumption the cell indices calculation for the particular sphere is done by the simple substitution of the sphere centre coordinates *X*[*ipart*], *Y*[*ipart*], and *Z*[*ipart*] into the cell coordinates *icell_x*, *icell_y*, *icell_z* respectively (see lines 67-72 and 369-371).

In conventional approaches to the link-cell method the length of the elementary cell side is larger than or equal to the cutoff radius. Since in the algorithm the cutoff radius depends on the current arrangement of the spheres and can be changed from one step to another, this assumption is no longer applicable. Thus in this case another approach is proposed in which the numbers of cells considered in the nearest neighbour searching are determined

in each particular case separately. They form a minimal cuboid containing all spheres, centres of which lie within the cutoff radius. Ranges of the cuboid are [*low_cell_x,lim_cell_x*], [*low_cell_y,lim_cell_y*], and [*low_cell_z,lim_cell_z*] respectively (see lines 67-72).

Given boundaries of the cells considered it is easy to check in what direction, if any, the periodic boundary condition (PBC) displacements [9] should be introduced. It allows to specify eight kinds of the most time-consuming loop for the nearest neighbour determination. In the first kind of the loop no PBC corrections are necessary (see lines 93-125). In the second one (lines 127-159) the perodic boundary conditions are considered in the z-direction only and so on, up to the 8-th loop kind (lines 331-363) in which the PBC corrections are aplied in the all three directions. In such an approach consisting of the eight kinds of the loop, increase of the overall program efficiency is achieved (see Table 3) due to careful elimination of the unnecessary operations for every kind of the loop separately.

The periodic boundary condition displacements are usually introduced using two conditional statements [4,9]. Contrary to the origin, in our program the PBC displacements are memorized in the *Pbc* table. Thus the conditional statements are replaced by the simple addition of the suitable *Pbc* element (for example see lines 347-349). The effectiveness of this modification can be seen in Table 3.

After relocation the coordinates of the current sphere *ipart_p* are changed. Thus it is necessary to update its position in the linked-list. For this purpose the sphere is removed from that list at the begining of the iteration. Then in lines 369-379 it is added again to the linked-list according to its current coordinates.

If no sphere center has been found within the cutoff radius the procedure ends at this point (line 383). Otherwise the current sphere is added to the sorted-list. In order to speed up the list handling a number of entry points to the sorted-list is introduced. The entry points are subsequently kept in the *Lsort_head* table. Thus *Lsort_head*[*idist*] is an entry point to the list of spheres from the related interval of distances to their nearest neighbours. *Lsort_list*[*i*] is the link coordinate for the sphere *i* and indicates the sphere with just greater distance to its nearest neighbour than *i*. Similarly to the linked-list, the sorted-list has its inverse table *Lsort_inv*. The value of *Near_neigh*[*i*] indicates the address of the coordinates of the nearest neighbour of the sphere *i*. The squared distance between the spheres *i* and *Near_neigh*[*i*] is stored in *Dist*[*i*].

The method of sphere relocation used may introduce new overlaps or may change or eliminate others. Thus the pair of intersecting spheres is not placed in the sorted-list if the overlap to which it corresponds could be changed by the elimination of the greater overlap. Subject to this restriction spheres are added to and removed from the sorted-list in every step of the simulation. This feature is reflected in lines 391-432 of the procedure code.

Tables 2 and 3 present some timings for runs of the program CPHS3D. They were obtained on a DSI-020 plug-in board for microcomputers compatible with IBM PC/AT or XT. The board consists of Motorola's 68020 CPU , 68881 hardware floating point coprocessor, 1MB of RAM, and clock 12.5 MHz. Another hardware applied was an IBM PC/AT - turbo (10 MHz) supported with a 80287 math coprocessor and MS DOS 3.10 operating system. The compilers used were : SVS C-compiler (version 2.6) for the DSI-020 board and Microsoft C-compiler (version 4.0) for the IBM PC/AT. For test purposes the program parameters were chosen in such a way that the final packing densities about 0.56 were obtained. In order to gain considerably larger densities however, much longer runs should be performed. The program run in which the packing density of 0.6499 was achieved took 13 days (!!!) of computer time on the DSI-020.

Table 2 is built on a base of a large number of runs performed in order to find the optimal division of the computational box into elementary cells. For each specified number of spheres the overall computational times were measured over the wide ranges of box partitions and next the minimal one was chosen. The ranges of numbers of cells considered varied from 4*4*4 to 32*32*32 for the DSI-020 and to 25*25*25 only for the PC/AT due to the Microsoft C-compiler restrictions concerning maximal arrays dimensions. From the tests described it clearly arises that the careful matching of the number of cells to the number of spheres highly influences the overall program efficiency. The values presented in the second column of the table show the total numbers of cells minimizing the overall computational time of the program. The third column indicates the almost linear relationship between the optimal number of cells and the number of spheres. One can also notice the almost constant program execution time per one sphere.

Table 3 shows average computer times per one sphere for four versions of the procedure *min_separation*(). They are averaged over runs with different numbers of spheres presented in Table 2 (in this case numbers of spheres ≥ 500 are considered only). Version 1 of the procedure is presented in Table 1. In version 2 the PBC corrections are considered by means of the two conditional statements instead of the *Pbc* table application. In the third version of the procedure the PBC corrections are memorized in the *Pbc* table as in version 1, but the only one case of the innermost loop is considered (lines 82-334 and

363-364 are removed from the procedure code). The fourth version stands for the base one and is originated from [4]. In this version the PBC corrections are introduced by means of the conditional statements (as in version 2) and only one case of the innermost loop is considered (similarly as in version 3).

Comparison of versions 3 and 4 shows that the PBC correction storage in the *Pbc* table results in about 10% reduction of the computational time. The similar effect is observed in a case of the innermost loop division (compare versions 2 and 4). Combining of those two techniques (version 1) gives the overall time savings of about 30%.

We are grateful to Prof. Roman Pampuch of the AGH for arranging a partial financial support for our activity under the project number CPBR 2.4.

Also we would like to thank to Professor E. M. Tory of the Mount Allison University who kindly supplied us with his program listing.

REFERENCES

[1] J.L. Finney, Nature 266 (1977) 309.
[2] W.S. Jodrey and E.M. Tory, Phys. Rev. A 32 (1985) 2347.
[3] W.S. Jodrey and E.M. Tory, Powder Technology 30 (1981) 111.
[4] W.S. Jodrey and E.M. Tory, Program source listing - private communication.
[5] M.H. Hughes and K.V. Roberts, Comp. Phys. Commun. 10 (1975) 167.
[6] J.P. Christiansen and K.V Roberts, Comp. Phys. Commun. 7 (1974) 245.
[7] R.W. Hockney and J.W. Eastwood, Computer simulation using particles (Mc Graw-Hill, New York, 1981) p. 48.
[8] W. Smith, "Fortran Code for the LINK-CELL Method" CCP5 Information Quarterly for Computer Simulation of Condensed Phases No. 20 (1986) 52 Daresbury Laboratory.
[9] D. Fincham, Comp. Phys. Commun. 21 (1980) 247.

Table 1. Source listing of procedure min_separation

```
 1                                                                    /*
 2  EXTERNAL VARIABLES
 3  ==================                                                */
 4                                                                    /*
 5  2. GEOMETRICAL PROPERTIES OF THE SYSTEM                           */
 6                                                                    /*
 7  2.1. PRINCIPAL GEOMETRICAL PARAMETERS                             */
 8
 9  extern double    *X, *Y, *Z;
10                                                                    /*
11  3. NUMERICAL SCHEME                                               */
12                                                                    /*
13  3.1. NUMERICAL PARAMETERS                                         */
14
15  extern int       Lsort_pos;
16  extern int       No_cells_1D, No_cells_2D;
17  extern int       No_rods;
18
19  extern double    Rcut2;
20  extern double    Scan1, Scan2, Shift;
21                                                                    /*
22  3.2. LINKED / SORTED LISTS                                        */
23
24  extern int       *Leap_cell_x;
25  extern int       *Leap_cell_y;
26  extern int       *Leap_cell_z;
27  extern int       *Link_head  ;
28  extern int       *Link_inv   ;
29  extern int       *Link_list  ;
30  extern int       *Lsort_head ;
31  extern int       *Lsort_inv  ;
32  extern int       *Lsort_list ;
33  extern int       *Near_neigh ;
34  extern double    *Dist       ;
35  extern double    *Pbc        ;
36                                                                    /*
37  END OF EXTERNAL DECLARATIONS
38  ===========================                                       */
39
40
```

```
41  min_separation(ipart_p)                                  /*  module c2s2  */
42                                                                            /*
43  2.2 CALCULATE MINIMUM SEPARATION OF ANY TWO SPHERES                       */
44
45  int     ipart_p;
46
47  {
48  int     inext, iprev, idist, ipart_near, jpart, ifirst;
49  int     leap_x, leap_y, leap_z;
50  int     icell_x, icell_xy, icell_y, icell_z, icell;
51  int     low_cell_x, low_cell_y, low_cell_z;
52  int     lim_cell_x, lim_cell_y, lim_cell_z;
53  int     idif_x, idif_y, idif_z, idif;
54
55  double pos_x, pos_y, pos_z, dist2, dist2_min;
56  double dif_x, dif_y, dif_z;
57                                                                            /*
58  ------------------------------------ MIN_SEPARATION - BEGIN ----- */
59                                                                            /*
60  1. SET UP INDEXING SYSTEM FOR SEARCH FOR OVERLAPS                         */
61
62      pos_x  =  X[ipart_p];
63      pos_y  =  Y[ipart_p];
64      pos_z  =  Z[ipart_p];
65
66      Near_neigh[ipart_p] = 0;
67      low_cell_x  =  pos_x + Scan1;
68      low_cell_y  =  pos_y + Scan1;
69      low_cell_z  =  pos_z + Scan1;
70      lim_cell_x  =  pos_x + Scan2;
71      lim_cell_y  =  pos_y + Scan2;
72      lim_cell_z  =  pos_z + Scan2;
73
74      ipart_near  =  0;
75      dist2_min   =  Rcut2;
76                                                                            /*
77  2. SET INDEX BASED ON COORDINATES
78
79     ONLY THOSE SPERES WHICH ARE CLOSE TOGETHER
80     ARE TO BE TESTED FOR OVERLAPS
81                                                                           */
82      idif_x  =  (Leap_cell_x[lim_cell_x] > Leap_cell_x[low_cell_x])
83                 ? 0 : 4;
```

```
 84      idif_y  =  (Leap_cell_y[lim_cell_y] > Leap_cell_y[low_cell_y])
 85                 ? 0 : 2;
 86      idif_z  =  (Leap_cell_z[lim_cell_z] > Leap_cell_z[low_cell_z])
 87                 ? 0 : 1;
 88      idif    =  idif_x + idif_y + idif_z;
 89
 90      switch (idif)
 91        {
 92
 93        case 0:
 94                                                                    /*
 95  2.1. NO PBC                                                       */
 96
 97          for (leap_x=low_cell_x; leap_x<=lim_cell_x; leap_x++)
 98              {
 99              icell_x = Leap_cell_x[leap_x];
100              for (leap_y=low_cell_y; leap_y<=lim_cell_y; leap_y++)
101                  {
102                  icell_xy = Leap_cell_y[leap_y] + icell_x;
103                  for (leap_z=low_cell_z; leap_z<=lim_cell_z; leap_z++)
104                      {
105                      icell = Leap_cell_z[leap_z] + icell_xy;
106                      jpart = Link_head[icell];
107                      while (jpart > 0)
108                            {
109                            dif_x  =  pos_x - X[jpart];
110                            dif_y  =  pos_y - Y[jpart];
111                            dif_z  =  pos_z - Z[jpart];
112                            dist2  =  dif_x * dif_x +
113                                      dif_y * dif_y +
114                                      dif_z * dif_z;
115                            if (dist2 < dist2_min)
116                               {
117                               ipart_near  =  jpart;
118                               dist2_min   =  dist2;
119                               }
120                            jpart  = Link_list[jpart];
121                            }
122                      }
123                  }
124              }
125          break;
126
```

```
127       case 1:
128                                                                    /*
129 2.2. PBC IN Z - DIRECTION                                          */
130
131         for (leap_x=low_cell_x; leap_x<=lim_cell_x; leap_x++)
132             {
133             icell_x = Leap_cell_x[leap_x];
134             for (leap_y=low_cell_y; leap_y<=lim_cell_y; leap_y++)
135                 {
136                 icell_xy = Leap_cell_y[leap_y] + icell_x;
137                 for (leap_z=low_cell_z; leap_z<=lim_cell_z; leap_z++)
138                     {
139                     icell = Leap_cell_z[leap_z] + icell_xy;
140                     jpart = Link_head[icell];
141                     while (jpart > 0)
142                           {
143                           dif_x  =  pos_x - X[jpart];
144                           dif_y  =  pos_y - Y[jpart];
145                           dif_z  =  pos_z - Z[jpart] + Pbc[leap_z];
146                           dist2  =  dif_x * dif_x +
147                                     dif_y * dif_y +
148                                     dif_z * dif_z;
149                           if (dist2 < dist2_min)
150                              {
151                              ipart_near  =  jpart;
152                              dist2_min   =  dist2;
153                              }
154                           jpart  = Link_list[jpart];
155                           }
156                     }
157                 }
158             }
159         break;
160
161       case 2:
162                                                                    /*
163 2.3. PBC IN Y - DIRECTION                                          */
164
165         for (leap_x=low_cell_x; leap_x<=lim_cell_x; leap_x++)
166             {
167             icell_x = Leap_cell_x[leap_x];
168             for (leap_y=low_cell_y; leap_y<=lim_cell_y; leap_y++)
169                 {
```

```
170                 icell_xy = Leap_cell_y[leap_y] + icell_x;
171                 for (leap_z=low_cell_z; leap_z<=lim_cell_z; leap_z++)
172                     {
173                     icell = Leap_cell_z[leap_z] + icell_xy;
174                     jpart = Link_head[icell];
175                     while (jpart > 0)
176                           {
177                           dif_x  =  pos_x - X[jpart];
178                           dif_y  =  pos_y - Y[jpart] + Pbc[leap_y];
179                           dif_z  =  pos_z - Z[jpart];
180                           dist2  =  dif_x * dif_x +
181                                     dif_y * dif_y +
182                                     dif_z * dif_z;
183                           if (dist2 < dist2_min)
184                              {
185                              ipart_near  =  jpart;
186                              dist2_min   =  dist2;
187                              }
188                           jpart  = Link_list[jpart];
189                           }
190                     }
191                 }
192             }
193         break;
194
195       case 3:
196                                                                        /*
197 2.4. PBC IN Y AND Z - DIRECTIONS                                      */
198
199         for (leap_x=low_cell_x; leap_x<=lim_cell_x; leap_x++)
200             {
201             icell_x = Leap_cell_x[leap_x];
202             for (leap_y=low_cell_y; leap_y<=lim_cell_y; leap_y++)
203                 {
204                 icell_xy = Leap_cell_y[leap_y] + icell_x;
205                 for (leap_z=low_cell_z; leap_z<=lim_cell_z; leap_z++)
206                     {
207                     icell = Leap_cell_z[leap_z] + icell_xy;
208                     jpart = Link_head[icell];
209                     while (jpart > 0)
210                           {
211                           dif_x  =  pos_x - X[jpart];
212                           dif_y  =  pos_y - Y[jpart] + Pbc[leap_y];
```

```
213                         dif_z  =  pos_z - Z[jpart] + Pbc[leap_z];
214                         dist2  =  dif_x * dif_x +
215                                   dif_y * dif_y +
216                                   dif_z * dif_z;
217                         if (dist2 < dist2_min)
218                            {
219                            ipart_near  =  jpart;
220                            dist2_min   =  dist2;
221                            }
222                         jpart  = Link_list[jpart];
223                         }
224                     }
225                  }
226               }
227         break;
228
229       case 4:
230                                                                        /*
231  2.5. PBC IN X - DIRECTION                                             */
232
233         for (leap_x=low_cell_x; leap_x<=lim_cell_x; leap_x++)
234             {
235             icell_x = Leap_cell_x[leap_x];
236             for (leap_y=low_cell_y; leap_y<=lim_cell_y; leap_y++)
237                 {
238                 icell_xy = Leap_cell_y[leap_y] + icell_x;
239                 for (leap_z=low_cell_z; leap_z<=lim_cell_z; leap_z++)
240                     {
241                     icell = Leap_cell_z[leap_z] + icell_xy;
242                     jpart = Link_head[icell];
243                     while (jpart > 0)
244                           {
245                           dif_x  =  pos_x - X[jpart] + Pbc[leap_x];
246                           dif_y  =  pos_y - Y[jpart];
247                           dif_z  =  pos_z - Z[jpart];
248                           dist2  =  dif_x * dif_x +
249                                     dif_y * dif_y +
250                                     dif_z * dif_z;
251                           if (dist2 < dist2_min)
252                              {
253                              ipart_near  =  jpart;
254                              dist2_min   =  dist2;
255                              }
```

```
256                         jpart  = Link_list[jpart];
257                         }
258                     }
259                 }
260             }
261         break;
262
263       case 5:
264                                                                    /*
265 2.6. PBC IN X AND Z - DIRECTIONS                                   */
266
267         for (leap_x=low_cell_x; leap_x<=lim_cell_x; leap_x++)
268             {
269             icell_x = Leap_cell_x[leap_x];
270             for (leap_y=low_cell_y; leap_y<=lim_cell_y; leap_y++)
271                 {
272                 icell_xy = Leap_cell_y[leap_y] + icell_x;
273                 for (leap_z=low_cell_z; leap_z<=lim_cell_z; leap_z++)
274                     {
275                     icell = Leap_cell_z[leap_z] + icell_xy;
276                     jpart = Link_head[icell];
277                     while (jpart > 0)
278                         {
279                         dif_x  =  pos_x - X[jpart] + Pbc[leap_x];
280                         dif_y  =  pos_y - Y[jpart];
281                         dif_z  =  pos_z - Z[jpart] + Pbc[leap_z];
282                         dist2  =  dif_x * dif_x +
283                                   dif_y * dif_y +
284                                   dif_z * dif_z;
285                         if (dist2 < dist2_min)
286                            {
287                            ipart_near  =  jpart;
288                            dist2_min   =  dist2;
289                            }
290                         jpart  = Link_list[jpart];
291                         }
292                     }
293                 }
294             }
295         break;
296
297       case 6:
298                                                                    /*
```

```
299  2.7. PBC IN X AND Y - DIRECTIONS                                    */
300
301          for (leap_x=low_cell_x; leap_x<=lim_cell_x; leap_x++)
302              {
303              icell_x = Leap_cell_x[leap_x];
304              for (leap_y=low_cell_y; leap_y<=lim_cell_y; leap_y++)
305                  {
306                  icell_xy = Leap_cell_y[leap_y] + icell_x;
307                  for (leap_z=low_cell_z; leap_z<=lim_cell_z; leap_z++)
308                      {
309                      icell = Leap_cell_z[leap_z] + icell_xy;
310                      jpart = Link_head[icell];
311                      while (jpart > 0)
312                            {
313                            dif_x  =  pos_x - X[jpart] + Pbc[leap_x];
314                            dif_y  =  pos_y - Y[jpart] + Pbc[leap_y];
315                            dif_z  =  pos_z - Z[jpart];
316                            dist2  =  dif_x * dif_x +
317                                      dif_y * dif_y +
318                                      dif_z * dif_z;
319                            if (dist2 < dist2_min)
320                               {
321                               ipart_near  =  jpart;
322                               dist2_min   =  dist2;
323                               }
324                            jpart  = Link_list[jpart];
325                            }
326                      }
327                  }
328              }
329          break;
330
331        case 7:
332                                                                       /*
333  2.8. PBC IN X, Y AND Z - DIRECTIONS                                 */
334
335          for (leap_x=low_cell_x; leap_x<=lim_cell_x; leap_x++)
336              {
337              icell_x = Leap_cell_x[leap_x];
338              for (leap_y=low_cell_y; leap_y<=lim_cell_y; leap_y++)
339                  {
340                  icell_xy = Leap_cell_y[leap_y] + icell_x;
341                  for (leap_z=low_cell_z; leap_z<=lim_cell_z; leap_z++)
```

```
342                     {
343                     icell = Leap_cell_z[leap_z] + icell_xy;
344                     jpart = Link_head[icell];
345                     while (jpart > 0)
346                           {
347                           dif_x  =  pos_x - X[jpart] + Pbc[leap_x];
348                           dif_y  =  pos_y - Y[jpart] + Pbc[leap_y];
349                           dif_z  =  pos_z - Z[jpart] + Pbc[leap_z];
350                           dist2  =  dif_x * dif_x +
351                                     dif_y * dif_y +
352                                     dif_z * dif_z;
353                           if (dist2 < dist2_min)
354                              {
355                              ipart_near  =  jpart;
356                              dist2_min   =  dist2;
357                              }
358                           jpart  = Link_list[jpart];
359                           }
360                     }
361                  }
362               }
363          break;
364        }                                          /*  switch - end  */
365
366                                                                    /*
367  3. ESTABLISH NEW CENTRE ipart_p IN ADDRESS GRID                   */
368
369      icell_x  =  pos_x;
370      icell_y  =  pos_y;
371      icell_z  =  pos_z;
372      icell    =  No_cells_2D * icell_x +
373                  No_cells_1D * icell_y +
374                                icell_z;
375      ifirst   =  Link_head[icell];
376      if (ifirst > 0) Link_inv[ifirst]  =  ipart_p;
377      Link_list[ipart_p]  =  ifirst;
378      Link_head[icell]    =  ipart_p;
379      Link_inv[ipart_p]   =  -icell;
380                                                                    /*
381  4. TEST FOR NEAREST NEIGHBOUR EXISTENCE                           */
382
383      if (ipart_near  ==  0) return;          /*  no nearest neighbour  */
384                                                                    /*
```

```
385  5. IN THE EVENT THAT THE POINT CLOSEST TO ipart_p IS ONE END
386     OF A ROD ALREADY QUEUED THE MAGNITUDE OF THAT ROD IS
387     COMPARED WITH THIS LATEST VALUE.
388     IF THE LATER IS SMALLER, THE 'OLD' ROD IS REMOVED FROM
389     THE QUEUE.
390                                                                    */
391      if ((jpart = Near_neigh[ipart_near])  !=  0)
392         {
393         jpart  =  (jpart > 0) ? ipart_near : -jpart;
394         if (Dist[jpart] <= dist2_min)
395
396             return;                    /*  no changes in sorted-list  */
397
398         Near_neigh[Near_neigh[jpart]] =  0;
399         Near_neigh[jpart]             =  0;
400         inext  =  Lsort_list[jpart];
401         iprev  =  Lsort_inv[jpart];
402         if (iprev > 0) Lsort_list[iprev] = inext;
403         else           Lsort_head[-iprev]= inext;
404         if (inext > 0) Lsort_inv [inext] = iprev;
405         }
406      else  No_rods++;
407                                                                    /*
408  6. MAP THE ROD INTO THE APPROPRIATE SORTED-LIST ENTRY POINT       */
409
410      idist  =  dist2_min * Shift;
411      if (idist < Lsort_pos) Lsort_pos = idist;
412      Dist[ipart_p]  =  dist2_min;
413                                                                    /*
414  6.1 FIND THE PROPER POSITION IN THE SORTED-LIST                   */
415
416      jpart  =  Lsort_head[idist];
417      iprev  =  -idist;
418      while (jpart > 0   &&  Dist[jpart] <= dist2_min)
419        {
420        iprev  =  jpart;
421        jpart  =  Lsort_list[jpart];
422        }
423      if (jpart > 0) Lsort_inv[jpart]  =  ipart_p;
424                                                                    /*
425  6.2 PLACE THE SPHERE IN THE SORTED-LIST                           */
426
427      Lsort_inv[ipart_p]      =  iprev;
```

```
428     if (iprev > 0)              Lsort_list[iprev]  =  ipart_p;
429     else                        Lsort_head[idist]  =  ipart_p;
430     Lsort_list[ipart_p]     =   jpart;
431     Near_neigh[ipart_p]     =   ipart_near;
432     Near_neigh[ipart_near]  =   -ipart_p;
433                                                                  /*
434 ------------------------------------ MIN_SEPARATION - END ------- */
435 }
```

Table 2. Typical Timings for CPHS3D Program[1].

No. of spheres	Optimal No. of cells	No. of cells per sphere	Computing time [s]		Computing time per sphere [s]	
			DSI-020	PC/AT	DSI-020	PC/AT
32	5* 5* 5	3.9	4	11	0.12	0.34
108	7* 7* 7	3.2	13	38	0.12	0.35
256	10*10*10	3.9	32	94	0.12	0.37
500	13*13*13	4.4	69	202	0.14	0.40
864	15*15*15	3.9	119	353	0.14	0.41
1372	18*18*18	4.2	192	565	0.14	0.41
2048	20*20*20	3.9	294	847	0.14	0.41
2916	22*22*22	3.7	420	1224	0.14	0.42
4000	23*23*23	3.0	582	1736	0.14	0.43

Table 3. Comparison of the CPHS3D time averages per sphere [s] for four versions of the procedure *min_separation*

	Procedure Version			
	1	2	3	4
DSI-020	0.14	0.15	0.16	0.20
PC/AT	0.41	0.44	0.45	0.60

[1]The presented times correspond to the final packing density of about 0.56.

Parallel Molecular Dynamics Algorithms on the Daresbury Meiko M10

W. Smith and D. Fincham

December 16, 1987

The present trend in MD simulations is to study progessively larger and more complicated systems. In such developments the limitations of scalar and even vector processors is recognised as seriously limiting the range of phenomena that can be investigated. There can be no doubt that the emergence of parallel processing offers the most promising means by which our aspirations may be realised. The presence of the Meiko computing surface and the FPS T/20 parallel processors at Daresbury, and the close association of the CCP5 project provides an excellent opportunity for the U.K. (at least) to participate in these exciting developments.

In the Advance Research Computing Group at Daresbury we have begun to develop new MD algorithms that will exploit the high degree of parallelism inherent in the physics of simulation. New algorithms are essential if the full potential of simulation is to be realised. Currently existing FORTRAN programs adopt well-known and well-tried techniques to optimise the efficiency and thus cost effectiveness of simulation. However these techniques have not viewed simulation from the perspective of inherent parallelism and consequently cannot necessarily be regarded as efficient in these kinds of application. In addition it must be recognised that exploitation of parallelism is not an integral aspect of FORTRAN and so it is necessary to look to newer languages such as Occam, which adequately express parallel concepts, to develop the new codes. Clearly a completely fresh look at simulation methods is required.

In the following article we describe our implementation of some simple parallelised "all pairs" molecular dynamics algorithms, suitable for simple Lennard-Jones liquid simulations, on the Meiko M10 computing surface. We do not say at this stage that these algorithms represent the best way of doing things, but they have interesting (and pedagogic) properties. In time we hope to develop a full repertoire of techniques in preparation for more advanced codes. We present two algorithms of the 'systolic loop' type [1]. The first of these, we call "Pass-the-parcel" (PTP) [1], in homage to the well-known children's game. It is a new algorithm for molecular dynamics and is very efficient. The second algorithm we call "Doubled-Pass-the-Parcel" (DPTP) for reasons which will emerge later. This algorithm can be couched in a form which is readily recognised as the parallel (SIMD) counterpart of the vector algorithm of

Brode and Ahlrichs [2,3].

1 The Meiko M10 Computing Surface

We begin with a brief description of the Meiko M10 (see Figure 1). It consists of 11 nodes comprised of a single T414 transputer and additional memory. Transputers are powerful microprocessors with the capability of being linked together to form parallel processing arrays [4,5]. Nine of these nodes are (in the Daresbury configuration) connected in a ring. The first of these nodes (p_0) is the "mass store" board with 8 M-bytes of memory. The other nodes in the ring ($p_1, \ldots, p_8$) have .25 M-bytes of memory each. These processors occupy two boards, and the system can be expanded by adding further boards. The mass store board also connects to a graphics board (G) and to a node called the "host" board (H). The host board is the node on which the editor, compiler and configurer programs reside and is the node which the user "talks" to via the front-end micro-VAX.

2 Pass-the-Parcel

In simulations of molecular liquids and solids it is often possible to dispense with neighbour lists and use a comparatively small system in which every molecule is considered to be a neighbour of every other molecule. Such an approach is valid provided the range of correlations present in the liquid is similar to the range of the potential.

Such "all-pairs" methods vectorise well, and, as we shall see, can easily adapt to parallel processing. It is advisable to use a computational cell which is as nearly spherical as possible (to avoid too many interactions which are outside the cut-off in the "corners" of the cell) and we have used truncated-octahedral or rhombic dodecahedral boundaries in our work. We have tested our approaches on Lennard-Jones argon, but there is no problem in principle in handling more complicated molecules.

For a system of N molecules there are N(N-1)/2 pair interactions to be evaluated, and we wish to use parallel processes to do this. Our discussion is in terms of processes rather than processors because parallel processes constitute one of the fundamental constructs within Occam, the native language of the Transputer. Parallel processes can be distributed over one or any number of processors with only minor changes to the program. Our first method, which we call Pass-the-parcel, is the simplest in the sense that there is only one copy of the coordinate data for each molecule present in the complete system of parallel processes. Since the evaluation of each pair interaction necessarily involves two molecules we can employ up to N/2 processes. In practice we take N odd and have (N-1)/2 parallel processes and thus can utilise up to (N-1)/2 processors. The way in which the method works is illustrated in Figure 2.

The (N-1)/2 identical Evaluator processes form a chain, with "upper" and "lower" communication channels able to pass data between them in

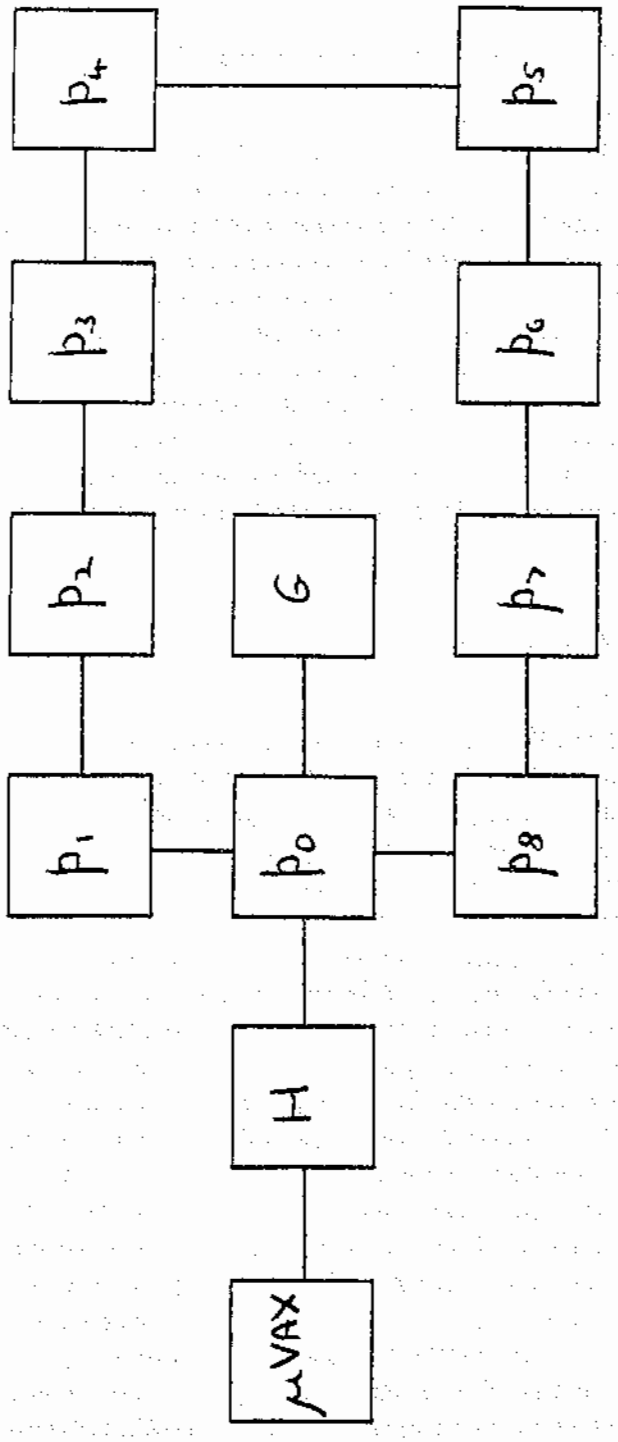


FIGURE 1. The Meiko M10 Configuration.

opposite directions. Processes Driver and Endchain at the two ends of the chain make the communication path into a closed loop. Each Evaluator works out the pair interaction between its "upper" and "lower" molecules, and respectively adds and subtracts the pair force onto the force accumulators for the two molecules. The coordinates and force accumulators for the molecules then circulate by one place in the manner shown in the Figure, and the next set of interactions is evaluated. After N steps all pair interactions have been considered, and each molecule returns to its "home" Evaluator, but with its force accumulator now complete. The molecule velocities have not been circulated, but remain in the home Evaluator. Each Evaluator can now integrate the motion of its own two molecules. During the pair interaction phase the only role of Driver is to circulate coordinates and forces (Endchain performing the same task at the other end), but now Driver has its own molecule to move.

The accumulation of thermodynamic quantities is done as follows. As each Evaluator calculates pair forces it also maintains a potential energy and virial accumulator. At the end of the step these accumulators, together with kinetic energies, are passed from left to right along the lower communication channels, summing as they go along. Endchain starts the process off by sending out zeros; Driver receives the grand totals and does the averaging and printing out. A velocity scaling factor is sent out from Driver to the Evaluators using the upper channels.

In principle PTP is best implemented on a chain of Transputers using two links between processors. As the Meiko is currently configured as a ring using single links, we placed both upper and lower channels on these links. Endchain and Driver then run on the same processor (p_0), which can communicate with the host Transputer, and hence to the outside world, by one of its other links. The Evaluator processes run on nodes $p_1, \ldots, p_8$, the number assigned to each depending on the number of molecules in the system. Because the program is written in terms of Occam parallel processes it is extremely simple to write and distribute over any number of processors. It does have an unfamiliar look to a conventional programmer: for example, there are no arrays anywhere in the program. The algorithm can be adapted for an even number of molecules, but it is probably simpler to add an extra ghost molecule if necessary.

There is a certain inefficiency in the way in which we do the accumulation of the thermodynamic quantities, as processes (and hence processors) must wait idly for the accumulators to arrive. It would be possible to improve this, but it is not worthwhile since this is such a small part of the calulation. We believe PTP is the most efficient possible systolic loop algorithm in terms of communication overheads, given the constraint of no data replication. In our case these overheads are negligible, and the speed-up using eight processors for the Evaluators is greater than eight. However the question of communication overheads may need to be reexamined for systems using large numbers of floating

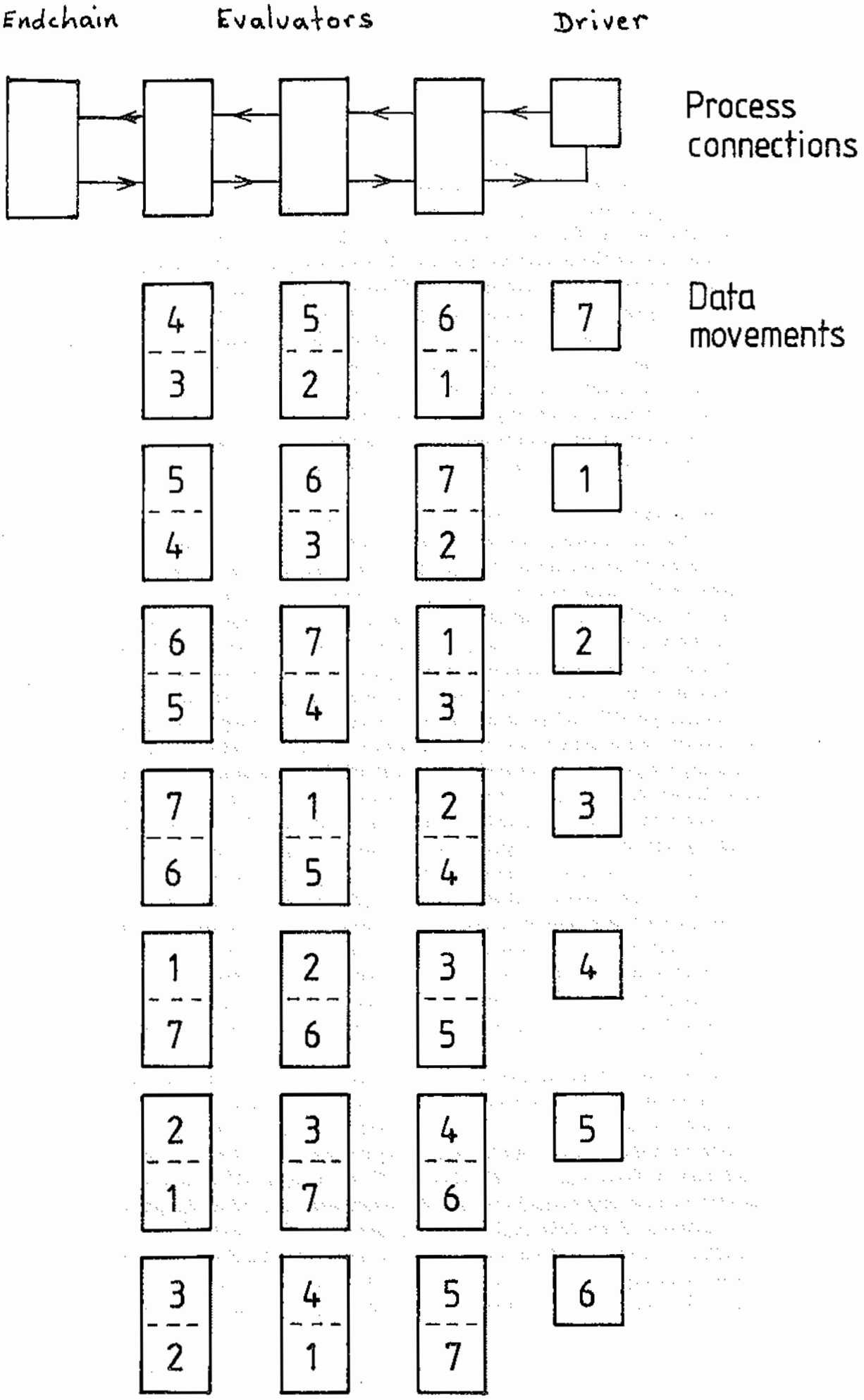


FIGURE 2. The Pass-the-Parcel Algorithm.

point (T800) Transputers.

In addition to the tasks already described Driver loads up the chain with initial coordinates, either from a file or by incorporating a lattice generator. It is an advantage of PTP that coordinates circulate through Driver in index order during the force evaluation phase, so it can write trajectory files in parallel with the computation. RDFs can be obtained during the simulation using separate histogram acccumulators in each processor, which are combined at the end of the run. VACFs can also be obtained during the run by means of history arrays for each molecule in its home Evaluator.

3 Doubled-Pass-the-Parcel

The Brode/Ahlrichs algorithm solves a rather awkward problem in vectorising molecular dynamics. Since it is necessay to calculate the interaction between pairs of molecules, we must perform a double-loop over the molecules to generate all possible pairs. In conventional molecular dynamics, we find that the lengths of the vectors being processed become progressively shorter as the forces are calculated (Figure 3) and thus the efficiency of the vector processing diminishes. Brode and Ahlrichs overcame this difficulty by restructuring the way the pairs are generated. The result being, that instead of dealing with (N-1) vectors of lengths ranging from (N-1) to 1, one deals with (N-1)/2 vectors of length N (Figure 4). (One may also append these vectors head-to-tail and obtain even longer vectors for maximum efficiency).

To couch this algorithm in a form suitable for parallel processing we have adopted, once again, the systolic loop method. However, unlike the PTP algorithm we now have TWO sets of accumulators associated with each molecule (hence the name "Double-PTP"). One set of accumulators remains "fixed" in a given parallel process, while the other is passed between parallel processes. The structure of the algorithm is as follows:

Consider a 7-molecule system. We may write the two sets of accumulators in rows thus:

1 2 3 4 5 6 7

2 3 4 5 6 7 1

In which the second row results from a cyclic shift of the second set of accumulators from their initial positions. (Thus the first row represents the fixed accumulators and the second row the moving ones.) Each column of number pairs represents a single pair-force calculation, and all of them may be calculated simultaneously, in parallel. Another cyclic shift produces the alignment:

1 2 3 4 5 6 7

3 4 5 6 7 1 2

and another set of parallel pair-force calculations can take place. Yet another cyclic shift produces:

1 2 3 4 5 6 7

4 5 6 7 1 2 3

which completes the entire set of pair force calculations for the 7-molecule

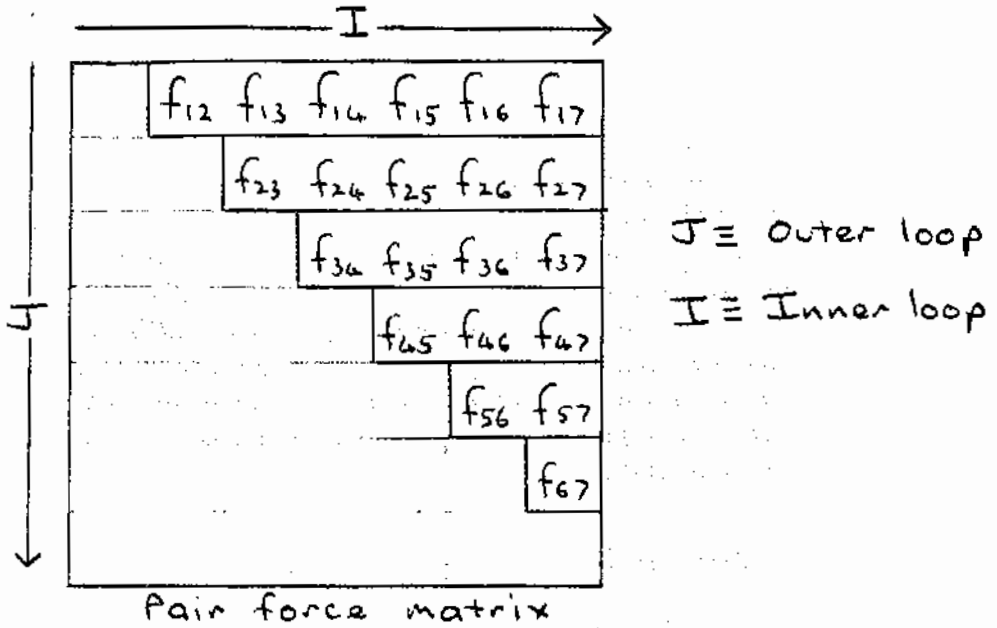


FIGURE 3. The Pair Force Matrix.

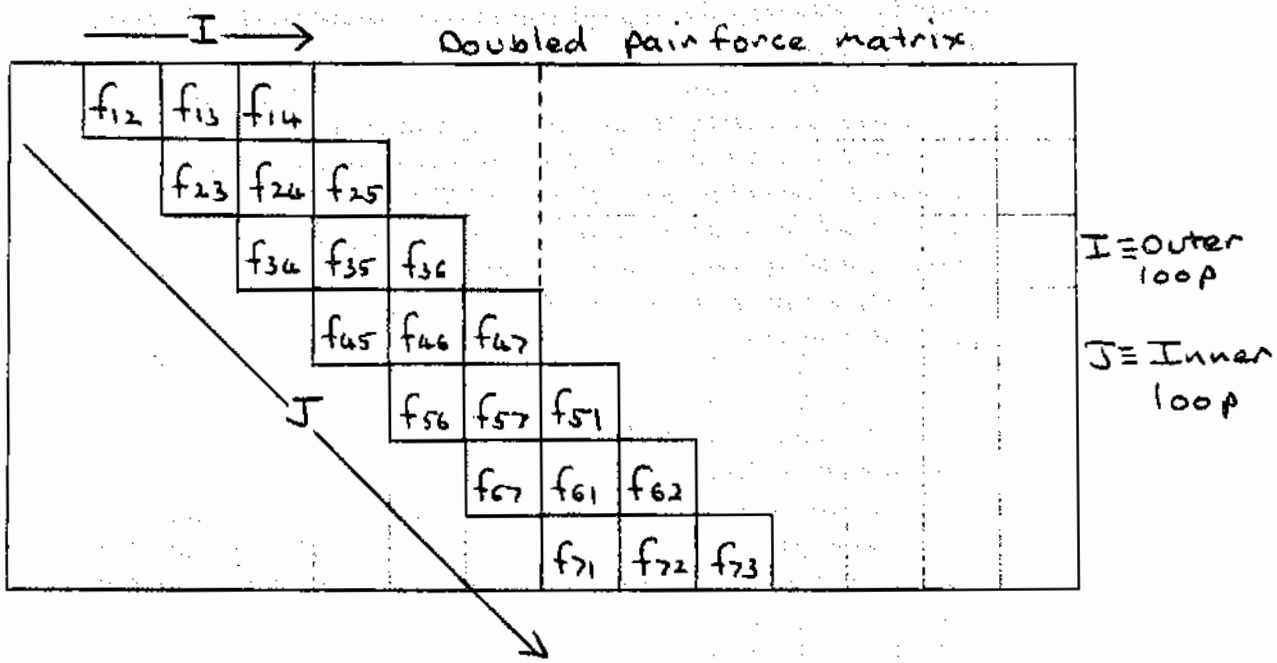


FIGURE 4. The Pair Force Matrix in the Brode-Ahlrichs Algorithm.

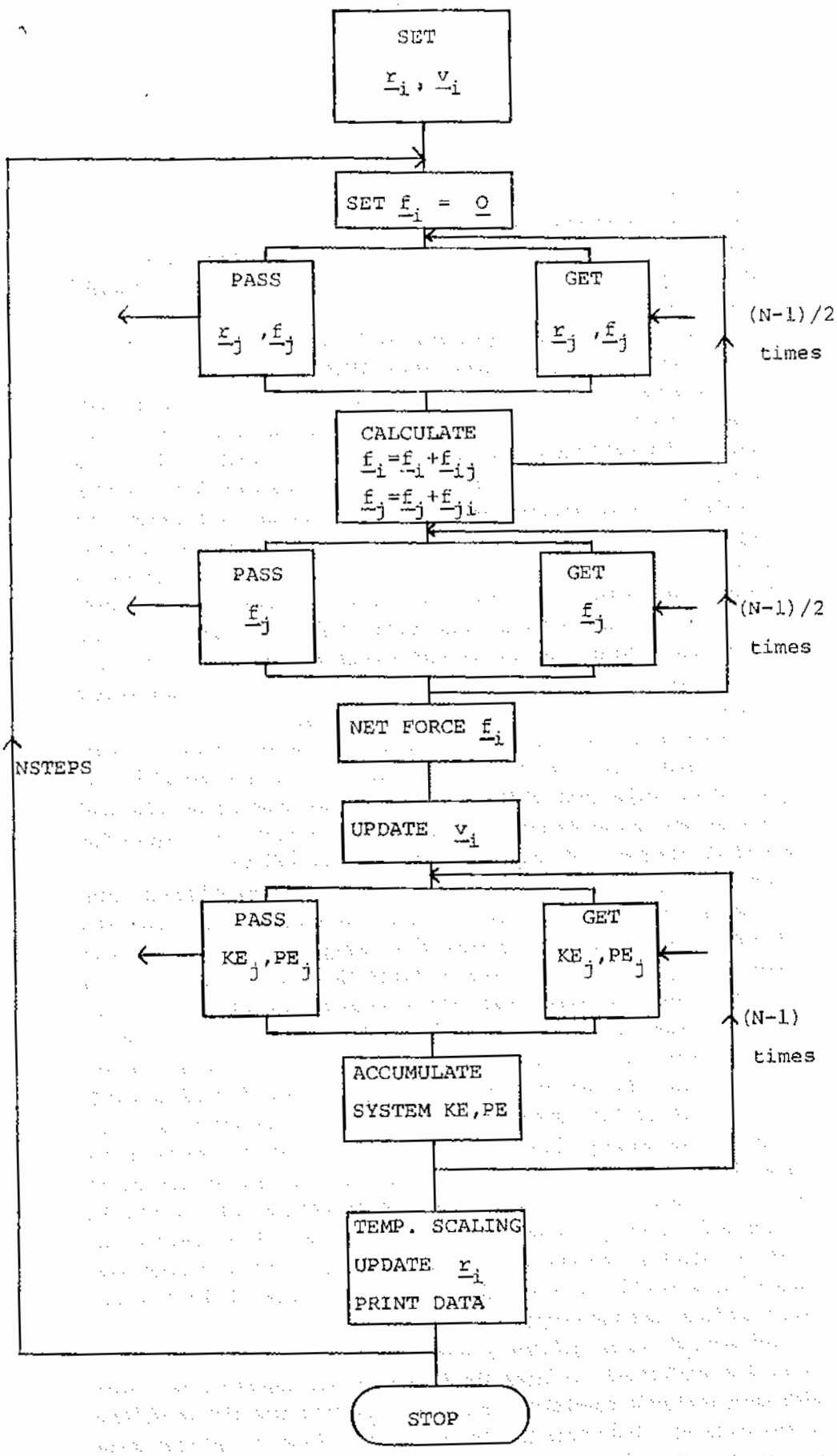


FIGURE 5 . ONE PROCESS FROM DOUBLED PASS-THE-PARCEL ALGORITHM

system. In general, for an N-molecule system, we should have N parallel pair-force calculations and (N-1)/2 cyclic shifts of the second set of accumulators to provide a complete calculation of all pair-forces in the system. This is to be compared with PTP, which has (N-1)/2 parallel processes and N cyclic shifts.

Certain features of this algorithm can be pointed out at once. It is apparent from the last row of pairs above that, after all the forces have been calculated, the two sets of accumulators for each molecule must me brought back together and added to give the full force acting on the molecule. This requires a "rewind" of the (N-1)/2 cyclic shifts. It is this rewind that makes the DPTP algorithm less efficient than the PTP algorithm. However, because the first set of accumulators remains fixed throughout the force calculations, we may identify a particular molecule with each parallel process. We could therefore make use of a maximum of N processors with this method. Finally we note that the cyclic shift operation is best visualised as a movement of the accumulators around a closed ring. These observations imply that the natural way to code this algorithm is as a set of N identical processes, running in parallel and passing data between each other in a ring topology.

The structure of the resultant program is presented in Figure 5. This figure represents one of the N processes that are to run in parallel. It is important to realise that all the processes in this program are identical (except in one detail described below) and this makes the algorithm particularly simple. A description of the program follows.

The first operation in each process is the initialisation of the molecule position and velocity. The position may be chosen randomly within the MD cell, provided that some means of reducing the effects of excessive overlap is employed. If this option is taken the velocity may be set to zero at the start. The rest of the code pertains to the MD cycle of force calculation, motion integration and thermodynamic data calculation.

In the first part of the MD cycle both sets of force accumulators are set to zero. The next operation is a loop of order (N-1)/2 in which the cyclic shift of data and the calculation of the pair force terms are performed sequentially. The data transferred between processes in the course of the cyclic shift are a molecule position vector and its associated force accumulator. An important aspect of this data transfer is that the sending and receiving operations occur **in parallel**. The need for this arises from the ring concept inherent in the design: if the operations were sequential, it would not be possible to complete a ring of simultaneous data transfers, and the program would "deadlock".

Following the force cycle the rewind of the force accumulators of order (N-1)/2 is performed, to bring the two force accumulators associated with each molecule together. These are added to give the total force on the molecule. Following this the molecule velocity is updated using a Verlet leapfrog algorithm. Note that, in each process, the motion of only one molecule is dealt with.

Next an (N-1)-fold loop moves the potential energy and kinetic en-

ergy calculated in each process around the ring. In each process the **total** energy is accumulated. There is an inevitable redundancy in this process, leading to N copies of the same thermodynamic data, but since these processes occur in parallel, no additional cost in time is incurred. The accumulation of the the kinetic energy in each process allows the calculation of the temperature and the appropriate temperature scaling performed (if required), without further communication of data between processes. The last part of the leap-frog algorithm may then be performed, namely; updating the molecule positions and application of the periodic boundary conditions.

The last operation of the MD cycle is the printing of the thermodynamic data. Since the same thermodynamic data is present in all processes, any of them may, in principle, perform this task. Practically the process nominated to do this must reside on the mass store node and is the only one that is in any way different from the others in the ring.

Implementation of this algorithm on the Meiko M10 is straightforward. The program is first developed (in Occam) to run as a ring of parallel processes on the "mass store" node of the network. (Of course this does not gain anything from parallelism, because the processes are merely time-sliced. However, the ability of the Transputer to do this makes it a very powerful development tool.) Once the program is debugged and working on a single node, it is relatively simple to spead the processes equally over the other nodes of the ring network. We still retain the feature of having several processes time-sliced on each node, and thus the algorithm is not optimally efficient. However, it makes up for this in terms of simplicity and flexibility; it is easily adapted to systems with greater or fewer processing nodes. (Of course, one should not rule out purchase of a few more transputers as the problem scales up!)

We are happy to report that spreading the processes over the 9 nodes of the Meiko M10 ring, results in at least a 9-fold increase in execution speed. (We say "at least" because, our experience indicates a slightly better ratio, due perhaps to the overheads associated with time slicing the entire job on a single transputer.)

References

[1] D. Fincham, "Parallel Computers and Molecular Simulation", Molecular Simulation **1** (1987) 1.

[2] S. Brode and R. Ahlrichs,"An Optimised MD Program for the Vector Computer Cyber 205", Comput. Phys. Comm. **42** (1986) 51.

[3] D. Brown, "Vectorising your MD Program for the Cyber 205 the Brode and Ahlrichs Way", CCP5 Information Quarterly, No. 24 (1987) p 39.

[4] D. Fincham, "The Transputer", CCP5 Information Quarterly, No. 22 (1986) p 51.

[5] S. Fornili, V. Martorana and M. Migliore, "The Transputer in Statistical Simulation", CCP5 Information Quarterly, No. 25 (1987) p 50.