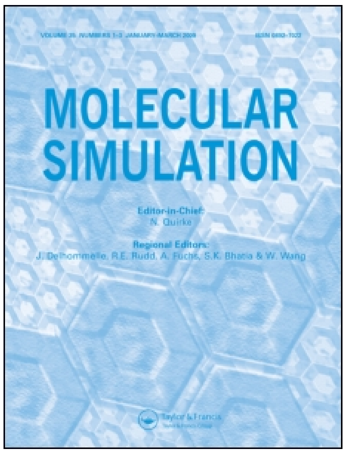

Molecular Simulation
Publication details, including instructions for authors and subscription information:
http://www.informaworld.com/smpp/title~content=t713644482

A short description of DL_POLY

W. Smith[a]; I. T. Todorov[a]
[a] Daresbury Laboratory, Computational Science and Engineering Department, CCLRC, Warrington, UK

To cite this Article Smith, W. and Todorov, I. T.(2006) 'A short description of DL_POLY', Molecular Simulation, 32: 12, 935 — 943
To link to this Article: DOI: 10.1080/08927020600939830
URL: http://dx.doi.org/10.1080/08927020600939830

Molecular Simulation, Vol. 32, Nos. 12–13, 15 October–15 November 2006, 935–943

A short description of DL_POLY

W. SMITH* and I. T. TODOROV

Daresbury Laboratory, Computational Science and Engineering Department, CCLRC, Daresbury, Warrington WA4 4AD, UK

(Received August 2006; in final form August 2006)

DL_POLY is a general purpose molecular dynamics simulation package with in-built parallel algorithms. It may be run on a wide selection of distributed memory parallel computers, from national supercomputers with thousands of processors, to single processor workstations and can simulate small systems with order 100 atoms, to systems with millions of atoms. This introduction provides an outline of the features of the package and the underlying methodology.

Keywords: DL_POLY; Molecular dynamics; Scientific software; Replicated data; Domain decomposition

1. Introduction

The DL_POLY molecular dynamics (MD) package has been under continual development at Daresbury Laboratory since 1994. It was first released to the academic community in 1996, so this special issue of Molecular Simulation marks its 10th anniversary as a public code. Its prime purpose on first release was to provide a simulation package for the UK CCP5 community [1] that was capable of exploiting the emerging parallel computers, of which the Intel IPSC 860 at Daresbury was a prime example. Since then it has gained popularity all over the world and is in considerable demand. It currently exists in two forms: DL_POLY_2, which is based on replicated data (RD) parallelism and DL_POLY_3, which is based on Domain Decomposition parallelism. Both versions are run on major parallel platforms all over the world.

DL_POLY was arguably the first public general purpose MD packages to be written specifically for parallel computers. Practical parallel platforms began to appear in the late 1980s and Daresbury was active in developing parallel algorithms at this early stage [2]. When the demand appeared for a new MD program for CCP5 we had already examined a number of possible strategies including RD [3], Systolic Loops [4] and Domain Decomposition [5,6]. The RD strategy was initially chosen because it offered the simplest approach to complex force fields, with reasonable scaling properties on platforms with up to 100 processors. The first version incorporating this strategy was developed in-house as DL_POLY_1 in 1994 and was circulated among close collaborators for testing and early exploitation. The first generally available package was eventually released in 1996 as DL_POLY_2 and was written by T. Forester and W. Smith. These codes were intended for *distributed memory* machines and this assumption underpins all DL_POLY codes to this day.

In the following sections, we outline the features available in the DL_POLY codes, including the force field specification, the parallel strategies that the codes are based on, techniques for modelling electrostatic interactions, the implementation of rigid bonds for parallel processing and the integration algorithms.

2. The DL_POLY force field

The DL_POLY package does not provide any particular set of force field parameters to describe the interatomic interactions, as with more specific packages such as AMBER [7], GROMOS [8] and CHARMM [9]. This is impractical given that the simulation code caters for widely disparate kinds of molecular system. It does, however, implement an enormous selection of functional forms for the interaction potentials arising in many of the force fields commonly used in molecular simulation. It is also easy, due to the structure of the software, for the user to add new potential functions and it should be noted that the user may use many different kinds of potential in the same simulation, which is not therefore confined to purely

*Corresponding author. Email: w.smith@dl.ac.uk

Lennard–Jones or purely Buckingham descriptions for example.

The total potential energy for DL_POLY is expressed by the formula:

$$
\begin{aligned}
V(\vec{r}_1, \vec{r}_2, \ldots, \vec{r}_N) = {} & \sum_{i,j}^{N'} U_{\mathrm{pair}}(r_{ij}) \\
& + \frac{1}{4\pi\varepsilon} \sum_{i,j}^{N'} \frac{q_i q_j}{r_{ij}} + \sum_{i,j,k}^{N'} U_{3-\mathrm{body}}(\theta_{ijk}) \\
& + \sum_{i,j,k,n}^{N'} U_{4-\mathrm{body}}(\phi_{ijkn}) \\
& + \left\{ \frac{1}{2} \sum_{i,j}^{N'} U_{\mathrm{FS}}(r_{ij}) + \sum_{i}^{N} F_{\mathrm{FS}}(\rho_i) \right\} \\
& + \frac{1}{2} \sum_{i,j}^{N'} \{ U_{\mathrm{Ter}}(r_{ij}) + \gamma_{ij} V_{\mathrm{Ter}}(r_{ij}) \} \\
& + \sum_{i_{\mathrm{bond}}}^{N_{\mathrm{bond}}} U_{\mathrm{bond}}(i_{\mathrm{bond}}, r_{ab}) \\
& + \sum_{i_{\mathrm{angle}}}^{N_{\mathrm{angle}}} U_{\mathrm{angle}}(i_{\mathrm{angle}}, \theta_{abc}) \\
& + \sum_{i_{\mathrm{dihed}}}^{N_{\mathrm{dihed}}} U_{\mathrm{dihed}}(i_{\mathrm{dihed}}, \phi_{abcd}) \\
& + \sum_{i_{\mathrm{invers}}}^{N_{\mathrm{invers}}} U_{\mathrm{invers}}(i_{\mathrm{invers}}, \psi_{abcd}) \\
& + \sum_{i=1}^{N} \Phi_{\mathrm{external}}(\vec{r}_i) \qquad (1)
\end{aligned}
$$

DL_POLY contains all the commonly used pair potentials ($U_{\mathrm{pair}}(r_{ij})$), including Lennard–Jones, Buckingham, 12–6, N–M and Morse potentials. The electrostatic interactions, indicated by the $q_i q_j / r_{ij}$ terms, are available as point charge and polarisable shell models, for which a variety of summation techniques may be selected (see Section 5). In DL_POLY polarisation is treated with the shell model of Dick and Overhauser [10] by the adiabatic method of Fincham [11] and the relaxation model of Lindan [12]. The three-body ($U_{3\text{-body}}(\theta_{ijk})$) and four-body ($U_{4\text{-body}}(\phi_{ijkn})$) interactions are non-specific angular potentials (suitable for glass simulations) and again offer a variety of functional forms. Many-body interactions, an increasing common requirement for modelling complex systems, are available in the Finnis–Sinclair form [13] for metals (terms $U_{\mathrm{FS}}(r_{ij})$ and $F_{\mathrm{FS}}(\rho_{ij})$) and Tersoff form [14] for covalent systems (terms $U_{\mathrm{Ter}}(r_{ij})$, γ_{ij} and $V_{\mathrm{Ter}}(r_{ij})$). All these mentioned forms are for non-bonded *inter*-molecular interactions.

For *intra*-molecular interactions DL_POLY has a wide selection of bond potentials (U_{bond}), angle potentials (U_{angle}), dihedral angle potentials (U_{dihed}) and inversion angle potentials (U_{invers}). The forms of these are taken from many published force fields including AMBER [7], GROMOS [8] and CHARMM [9] and Dreiding [15].

Lastly, DL_POLY permits the user to implement external force fields. This capability is useful for modelling transport (e.g. conduction), or containment (e.g. pores) or mechanical intervention (e.g. shearing).

3. Integration algorithms

The integration algorithms in DL_POLY handle the dynamics of the system being simulated. From the current positions of the atoms, the forces may be calculated from the first derivatives of the potential functions outlined above and used to update the atomic velocities and positions. The integration progresses in a sequence of finite steps in time, each time step being of the order 1 ~ 10 fs. The algorithms for this purpose in DL_POLY are based on the Verlet leapfrog (LF) and velocity Verlet (VV) schemes [16].

In addition to providing a numerical solution to the equations of motion, the integration algorithm also defines the thermodynamic ensemble. At the base level, both LF and VV provide the NVE (constant energy ensemble), but we have also implemented in DL_POLY the NVT (canonical) ensembles of Evans [17], Hoover [18] (after Melchionna *et al.* [19]) and the pseudo canonical ensemble of Berendsen [20]. For constant pressure work the isotropic isothermal-isobaric (NPT) ensemble has available in both Hoover and Berendsen forms and complemented by the anisotropic forms (NST) for simulation of phase transitions in solids.

As well as the integration/ensemble algorithms DL_POLY also accepts molecular structures defined by rigid bonds and, in the case of DL_POLY_2 only, rigid bodies. The types of molecular structures that may be accommodated in a DL_POLY simulation are shown in figure 1. It is important to note that all such structures may be present in one simulation!

Rigid bonds adapt easily within the framework of LF and VV, though the well known algorithms SHAKE (LF) [21] and RATTLE (VV) [22] and we have devised versions of these for DL_POLY that are both parallel and appropriate for the above ensembles. Considerations pertaining to the parallel SHAKE algorithm are described in Section 6.

Rigid bodies may be used to represent structures like aromatic hydrocarbons and their derivatives, which arise in all branches of chemistry. In DL_POLY_2 the dynamical treatment of such entities is based on Euler's prescription [23] augmented by a quaternion treatment of the orientation [24]. For the LF integration scheme

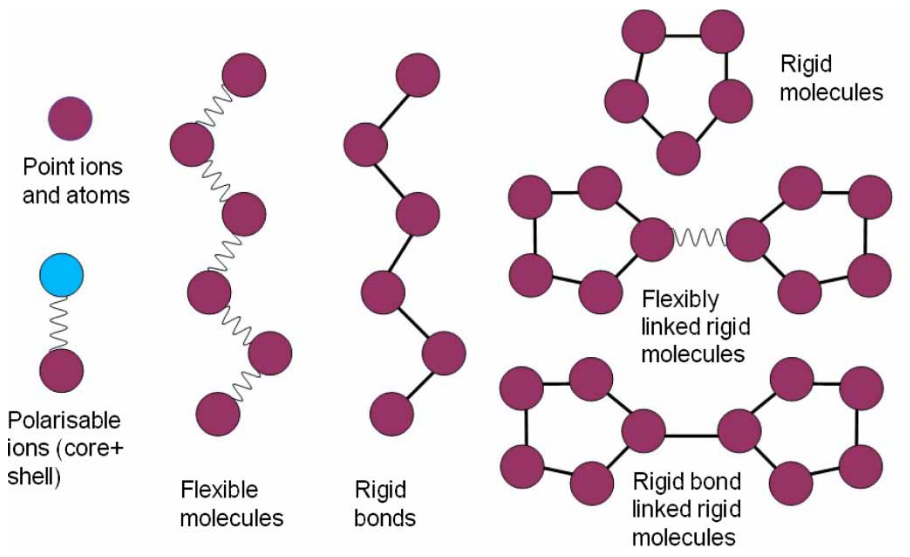
Point ions and atoms
Polarisable ions (core+ shell)
Flexible molecules
Rigid bonds
Rigid molecules
Flexibly linked rigid molecules
Rigid bond linked rigid molecules

Figure 1. Molecular structures supported by DL_POLY. Any or all of such structures may be present in a given model at the same time. Rigid bodies however (right of diagram) are not available in DL_POLY_3.

DL_POLY_2 employs the Fincham implicit quaternion algorithm [25] and for the VV scheme the NOSQUISH algorithm of Miller *et al.* [26] is used. The latter algorithm has the advantage of being symplectic and therefore stable for long time integrations [26].

The presence of both rigid bonds and rigid bodies in the same systems raises the possibility of rigid bodies linked by rigid bonds. A suite of integration routines are available for this situation in DL_POLY_2. These routines are derived from the QSHAKE algorithm [27] which was devised by us and is able also to generate any of the ensembles described above.

4. Parallelisation strategies

4.1 Replicated data parallelism

In the RD strategy [3] each processor of the parallel machine maintains a replica of the configuration of the simulated system, i.e. the coordinates $\{\underline{r}_i\}$, velocities $\{\underline{v}_i\}$ and forces $\{\underline{f}_i\}$ for all atoms $\{i = 1, \ldots, N\}$ in the system. Each processor may be thought of as running the same simulation, but replication of the computational effort is avoided by assigning to each processor a subset of the tasks involved. The simulation as a whole is established by communicating the results of these concurrent tasks to all processors, so that every processor can continue to maintain a full replica of the simulation. The communication of these data is inevitably a global operation, since all processors need all the data. Key points at which this global operation is necessary are in the computation of forces and in the integration of the equations of motion.

The principal expense in MD lies in the computation of atomic forces and this is where most of the effort lay in developing DL_POLY. Calculation of *intra*-molecular forces is handled though bookkeeping arrays that store the identities of interacting atoms and the relevant force forms (bonds, angles, dihedrals, etc.). Parallelism is achieved by simply allocating each processor a subset of bookkeeping arrays (figure 2). The overall efficiency of this approach is extremely high. The basic approach is suitable for both RD and DD implementations, though there are additional complications with DD (see Section 4.2).

A more difficult task is the parallel distribution of *inter*-molecular forces, of a type similar to van der Waals (VDW) interactions. The approach adopted is to construct a distributed Verlet Neighbour List [16] based on the Brode-Ahlrichs decomposition of the pair force matrix [28], which is built independently on each processor so that no pair interaction is replicated (figure 3). This is almost ideal parallelism and therefore highly efficient,

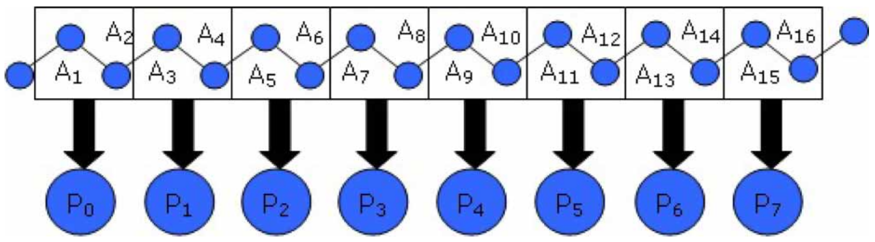

Figure 2. Division of intramolecular force terms over processors. Here, bond angle terms A_1–A_{16} are evenly allocated to specific processors for computation.

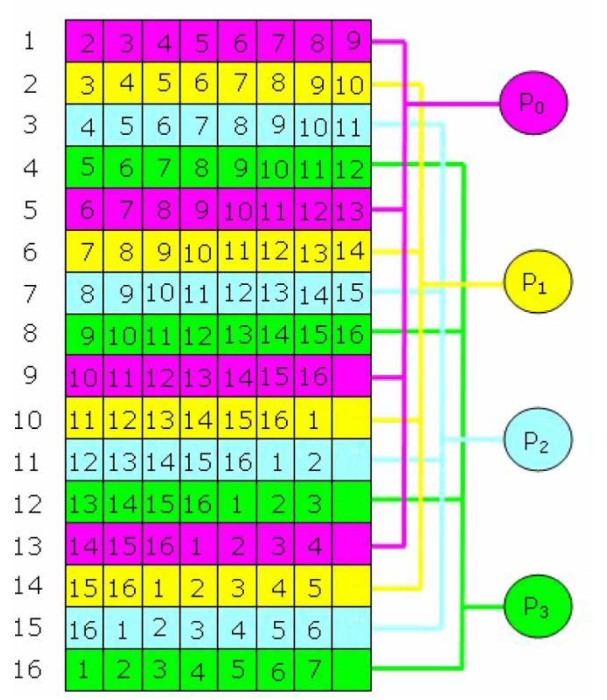

Figure 3. The parallel distribution of the Verlet neighbour list in DL_POLY_2 on a 4 processor machine. The pair force matrix is restructured according to the scheme of Brode and Ahlrichs and rows are assigned to processors P_0–P_3 according to the colour scheme, to achieve a reasonable load-balance across processors.

since each processor can build a list without communicating with others, and the resulting list is a fraction of that for the whole system. In keeping with the Verlet method, the list must be updated at intervals during the simulation, which is accomplished by monitoring the distances atoms move and updating the list when a threshold is exceeded. The neighbour list enables efficient identification of interacting atom pairs. Intrinsic to our implementation is a strategy for omitting VDW interactions from the neighbour list if the atom pair concerned is already part of an *intra*-molecular interaction.

For more complicated intermolecular interactions, such as the Finnis–Sinclair [13] or Tersoff [14], which are density dependent, DL_POLY_2 employs a Link Cell [29] approach. More problematical however, is the treatment of long ranged forces via the Ewald method [30]. This merits special consideration and is described in Section 5.

It is apparent that distribution of the force calculations under RD results in a partial description of the full force field on any one processor. Thus a global communication step is required to establish full replication of the force data everywhere. In DL_POLY_2 this is accomplished by a global summation of the atomic force arrays. This is a relatively expensive step in communication terms, and is a principal reason why the RD strategy is not recommended for computers with high processor counts. However, experience shows that the impact of this step is very dependent on the nature and size of the simulation being undertaken and simulations that are large (approaching say 30,000 atoms) are known to scale quite well, sometimes up to 250 processors with good communication hardware.

The second opportunity for exploiting parallelism under RD occurs in the numerical integration of the equations of motion. Thus in DL_POLY_2 each processor integrates the motion for a subset of atoms only. Replication of the coordinate and velocity arrays is established once again by a global communication using systolic loops. Integration of the equations of motion frequently involves employing the SHAKE algorithm for systems with rigid bonds [21]. This is an issue which is dealt with in Section 6.

Though DL_POLY_2 proved itself beyond its original design in terms of system sizes and processor counts, it became apparent in the early 2000s that the underlying RD strategy was not appropriate for the emerging platforms with thousands of processors, so work began on a DD version that would scale more efficiently on such platforms. This eventually became the DL_POLY_3 code that appeared in 2004 and was written by I. T. Todorov and W. Smith [31].

4.2 *Domain decomposition parallelism*

The DD strategy is radically different from RD. Under the DD approach the simulation cell is divided spatially into quasi-independent domains which are allocated to individual processors.

If follows immediately that the simulated system must be reasonably isotropic if a reasonable degree of load balancing amongst the processors is to be achieved. The spatial division naturally does not recognise molecular entities, which are therefore usually divided between processors, creating special communication difficulties. The implementation of DD in DL_POLY_3 is based on Hockney and Eastwood's Link Cell algorithm [29], which was adapted for parallel use by Pinches *et al.* [5] and Rapaport [6]. A Link Cell approach is not entirely essential for DD, but it provides useful constructs to aid its implementation and yields order N scaling for large numbers of atoms, N. The structural aspects of DD are shown in figure 4.

Spatial partitioning for DL_POLY_3 demands that the number of processors P must have the form $P = 2^n$ where n is an integer. This is a requirement arising from the Ewald calculations described in Section 5.2. The MD cell is most often divided into near-cubic domains, though exception is made for systems with slab geometries to help achieve load balance. Each domain is then sub-divided into link cells according to the normal prescription, in which the width of a link cell must be greater than the cut-off distance applied to all *inter*-atomic interactions. (In the context of DL_POLY_3, this criterion must also include the 1–4 distance in the *intra*molecular dihedral potentials.) Ideally, these requirements should lead to better than a $3 \times 3 \times 3$ link cell partitioning of the domain in the three principal directions. DL_POLY_3 can handle fewer link cells per domain than this, but it raises major

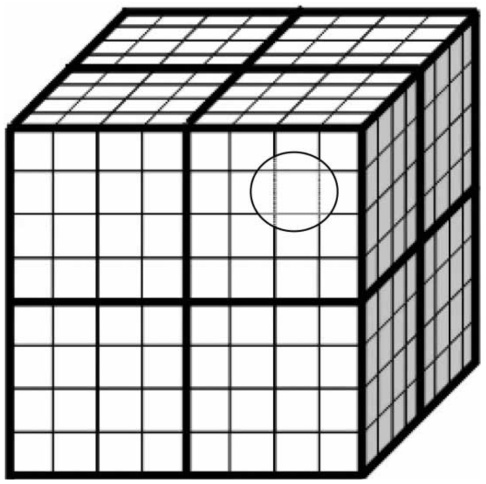

Figure 4. Domain Decomposition. The MD cell (large cube) is divided into equal domains (middle cubes) each of which is allocated to a specific processor. Each domain is divided into link-cells (small cubes) the width of which must be greater than the radius of cut-off applied to the inter-atomic force terms. The sphere above represents the cut-off sphere defining the interaction range.

efficiency issues arising from the construction of the "halo data".

The "halo data" represents the construction around each domain of a partial image of all neighbouring domains so that calculation of all the forces relevant to a domain can take place (figure 5). In DL_POLY_3 this amounts to the transfer of the atomic coordinates of all atoms located in link cells at the boundaries of a domain to the processors managing the neighbouring domains. This is a six-fold transfer operation that moves data in directions North, South, East, West, Up and Down of each domain. These six transfers do not happen concurrently, since some data sorting is necessary to populate the "corners" of the halo data. It is apparent from the nature of the link cell method, that these transfers are sufficient for a complete calculation of the forces on all atoms in any domain. It is also apparent that if the domains have relatively few link cells (or their shape is far from cubic), then the transfer of the halo data represents the transfer of a major proportion of the contents of a domain, which implies a large, possibly prohibitive, communication cost. This can be avoided by running the program on fewer processors.

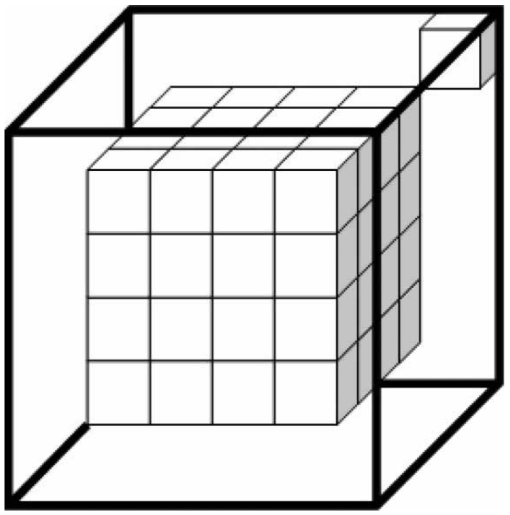

Figure 5. Halo data construction in Domain Decomposition. The central cube represents a spatial domain that is allocated to a single processor, where it is divided into link-cells (small cubes). Surrounding the domain it is necessary to add the halo data, which is one link-cell in width (indicated by the isolated small cube), as it is composed of the coordinates of atoms found in the link-cells at the boundaries of the neighbouring domains. It is apparent from this construction that the smaller the link cells, the more efficient will be the overall algorithm, since less data will need to be transferred.

The transfer of the halo data is the main communication cost of the basic DD strategy. After the transfer, the atomic forces may be calculated and the equations of motion integrated independently on each processor. Atoms that move sufficiently far may then be reallocated to a new domain.

Computation of *intra*-molecular forces can be accomplished, in principle, using the partitioning scheme described in figure 2. However, there are particular complications arising from the DD scheme. There are two aspects to this: firstly the description of the molecular structures (commonly called the topology) is "broken" by the decomposition into domains; and secondly the evolution of the system demands that the topology be partially reconstructed every time atoms move from one domain to another. In order to accomplish this, the package of data transported with each atom that leaves a domain contains not only its configurational data (position, velocity and force), but also a topological description of the bonding terms associated with the atom.

5. The treatment of long ranged electrostatic forces

5.1 The standard Ewald sum

The treatment of long ranged electrostatic forces represents a particular challenge in molecular simulation. Direct summation of the Coulomb pair interactions is rarely adequate, except for the treatment of atomic clusters, so more sophisticated treatments have evolved. The main methods used in DL_POLY are based on the Ewald sum [30].

The Ewald sum casts the sum of Coulomb pair interactions into two separate sums (plus a correction term, which is computationally trivial). The first sum is a screened Coulomb sum, which resembles the Coulomb formula but each term is weighted by a screening function (the complementary error function *erfc*) which compels the sum to converge in a finite range. The second sum is a sum of structure factors, which are calculated from reciprocal space vectors, and which are again weighted by a screening function (this time a Gaussian) which guarantees a finite sum. The first sum is therefore set in *real* space, while the second is set in *reciprocal* space. The convergence of both sums is governed by a single parameter α, which defines the range of both convergence functions and is known as the Ewald convergence parameter.

The original implementation of the Ewald sum in DL_POLY_2 was a RD adaptation [32] devised by us. Later this was augmented by a similar adaptation of the Hautman–Klein–Ewald (HKE) method [33] for systems with 2D periodicity, and a partially distributed adaptation of the Smoothed Particle Mesh Ewald (SPME) [34]. A fully distributed SPME version was implemented as the primary method in DL_POLY_3.

The RD adaptation of Ewald's method [32] requires no special modifications for calculating the *real*-space components. These are treated in the same way as the VDW terms described above. The *reciprocal*-space terms, which are derived from a Fourier transform of the system charge density, are parallelised through an atomic decomposition: each processor is made responsible for a fixed set of atoms. The method involves the global summation of the structure factors associated with each reciprocal space vector. The cost of this was minimised in later versions by summing the structure factors for all vectors simultaneously. The same approach is taken with the HKE method.

5.2 The smoothed particle mesh Ewald sum

The RD adaptation of the SPME method employs the same treatment of the *real*-space terms as for the standard RD Ewald approach. However the key difference is in the treatment of *reciprocal*-space, which is an interpolation of the charge distribution, based on Cardinal B-splines [33], on a regular 3D grid. This permits the use of a 3D Fast Fourier Transform (FFT) to calculate the structure factors, which accelerates the process enormously. An important consideration is how to parallelise the method. Distributing the central FFT operation risks impairing the supreme efficiency of the FFT algorithm. In DL_POLY_2 the decision was made to replicate the full FFT operation on all processors, though the construction of the 3D charge array needed for the calculation is constructed in a distributed fashion and completed by a single global sum operation. This strategy is inevitably expensive in memory terms but is simple to implement and does not disrupt the FFT algorithm, which therefore retains its efficiency. The method has proved to work with acceptable efficiency, scaling reasonably well for large simulations and processor counts of order 256.

For the DD approach in DL_POLY_3, a fully distributed implementation of the SPME method was essential. This was accomplished through a distributed 3D FFT algorithm devised by Bush [35]. Known as the Daresbury Advanced Fourier Transform (DAFT), this FFT employs a domain decomposition of the 3D FFT arrays which maps neatly on to the DD structure of DL_POLY_3. This means that all computations necessary to build the (partial) arrays can take place without inter-processor communication. Furthermore all communication required by the FFT algorithm is handled internally. While the insertion of communication processes into the heart of the FFT algorithm inevitably affects the efficiency of the FFT calculation, DAFT nevertheless possesses excellent scaling characteristics and the associated economies in data management resulting from its use makes the DL_POLY_3 SPME implementation a highly efficient algorithm [36]. Radiation damage simulations of order 2 million atoms (and larger) are regularly performed with this program [37].

6. The parallel SHAKE algorithm

An essential requirement for all MD codes intended for modelling complex systems is a means of handling rigid interatomic bonds (also called constraint bonds). Not only is this necessary to permit practical (i.e. not too short) time step intervals, but also to remove the problem *nonergodocity* (poor coupling of the system degrees of freedom) that delays, or even prevents, the onset of equilibrium. Both versions of DL_POLY implement the SHAKE algorithm for rigid bonds [21].

The principles of the SHAKE algorithm are well known. In the first stage the motions of atoms are integrated without consideration of the rigid bonds. In the second stage, a correction to the displacement of the atoms is applied to restore the required bond length. The correction is applied to each bond in turn and is applied iteratively, so that perturbations to each bond due to corrections applied to neighbouring bonds may be allowed for. The iteration ceases when all the bonds have converged, to within a tolerance of order $\sim 10^{-5}$ of the required length.

A parallel implementation of the SHAKE algorithm introduces new considerations. In order to distribute the work load over a number of processors it is sensible to allocate independent sets of constraint bonds to different processors. Since atoms are generally linked into more than one bond by virtue of their valency, this inevitable means that some bonded atoms will be handled by more than one processor. It follows (figure 6) that during the iterative stage of SHAKE, it is necessary to communicate between processors to ensure that the corrections applied to individual atoms take account of the full connectivity of the molecular structure. This problem was first solved by the RD_SHAKE algorithm [38] which was intended for RD implementation, but the techniques can be carried over to DD. In fact, the communication overhead in DD is much less than in RD, since data replication at intermediate stages is not required.

The RD implementation of SHAKE therefore requires firstly that the rigid bonds for which the processor is responsible be identified and a list compiled of the number of such bonds each atom participates in. This list is circulated to all other processors, from which each processor may establish a "shared atom" list which records which atoms are shared with which processor. Then during the SHAKE iteration, changes in the positions of shared atoms are communicated to the appropriate processor, thus avoiding the problem indicated

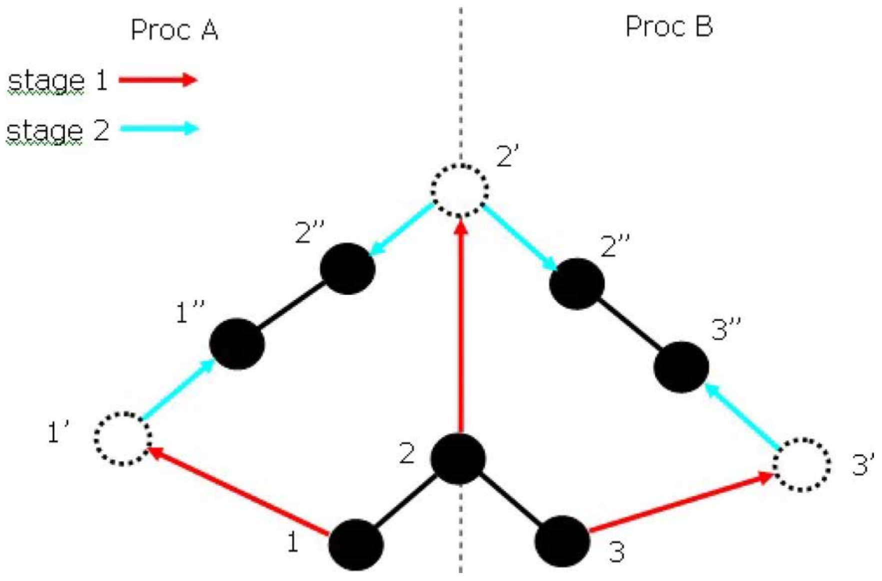

Figure 6. The SHAKE algorithm on multiple processors. Atoms 1,2 and 3 represent part of a molecular structure. The bond 1–2 is handled by processor A and bond 2–3 is handled by processor B. Stage 1 of the shake algorithm will carry atom 1–1′, 2–2′ and 3–3′. Stage 2 iterates these positions until the final positions 1″, 2″ and 3″ have conserved bond lengths. It is apparent however, that unless processors A and B communicate during iteration, atom 2 will relax to different positions on the two processors.

in figure 6. In the RD case the list of shared atoms needs to be compiled only once, at the start of the simulation. For DD implementation however, the shared atom list must be continually updated during the simulation, as atoms move between processors. Fortunately, this is purely a data transfer issue and does not involve a repeat of the reconstruction of the complete shared atom arrays.

Similar to SHAKE, the RATTLE algorithm was also developed to treat constraints, but in the VV scheme. It has two parts: the first is an iterative correction to the constrained atom positions as in SHAKE; and the second is an additional iteration procedure to constrain atom velocities so that the component of their relative velocity along the constraint bond is zero, within a given tolerance. Both parts are similar in implementation to the RD SHAKE scheme.

7. General comments

7.1 Extension of the code

As described above, DL_POLY_2 and DL_POLY_3 contain a common set of functionality despite of their differences in strategies of parallelisation. (Though we emphasise again that certain aspects of molecular topology, e.g. rigid bodies, present considerable difficulties in the DD implementation and are therefore not currently available.) However, each of the packages also contains unique features which were developed in response to demands from users. Thus DL_POLY_3 supports defect detection tools and a variable timestep algorithm, suitable for highly non-equilibrium simulations such as radiation damage studies (with correspondingly large systems), which require such features. More information about these can be found in the respective manuals [39,40]. However, our vision for the future is to keep these two packages as mutually compatible as possible.

The user community is encouraged to extend the current functionality of the packages for their own benefit. For this purpose the software is supplied in source form. Ideally such modifications would find their way back into the standard versions of the programs, but they must comply with the high standards of coding and documentation of the overall package. Unfortunately we can make no commitment to verify, extend or develop extensions contributed by users, but at our discretion we may develop certain contributions if we judge they would benefit the wider DL_POLY community.

7.2 Porting issues

The DL_POLY packages are written in highly modularised FORTRAN 90 and do not make use of any external libraries. Thus the packages are fully self contained. Users can, of course, substitute the DL_POLY FFT routines with local or vendor specific versions at compile time, though we do not recommend this for DL_POLY_3 on account of the unique matching of the 3D FFT to the DD force calculations.

The packages are supplied with template makefiles to handle assembly and compilation in serial or parallel modes in either UNIX compatible or UNIX emulated environments (e.g. Windows with CygWin [41]). Ideally users would find these makefiles entirely sufficient, but a working knowledge of compiler flags and optimisation issues is an advantage. While compilation in serial mode is straight forward, parallel mode compilation raises additional considerations. Inter-processor communications in DL_POLY are implemented through FORTRAN MPI calls. For successful compilation and flawless execution users must ensure that their communication hardware has stable builds of the MPI software compiled with respect to the corresponding FORTRAN 90 compiler.

7.3 The DL_POLY Java GUI

The DL_POLY suite also features a basic GUI for managing some aspects of code use. Written in Java, and therefore highly portable, the GUI can help with construction of input data, job submission and analysis of the simulation output. Given the high degree of versatility of DL_POLY it has not been possible to develop a GUI that satisfies every user need. For those willing to experiment however, it can prove a valuable aid in exploiting DL_POLY. The GUI is supplied with the DL_POLY source and is fully documented [42].

7.4 Obtaining DL_POLY

The DL_POLY programs are available free of charge (to academic researchers) from the DL_POLY website [43]. The software is supplied under a licence protecting the commercial rights of Daresbury Laboratory and the potential user must agree to these terms before downloading the source.

8. Conclusion

The DL_POLY package provides a powerful and versatile set of programs for MD simulation of complex molecular systems. The potential range of applications is vast, as this issue of Molecular Simulation demonstrates. Systems as small as 100 atoms and as large as 30 million atoms can be simulated with it. The code is free (to academics) and the source is open to inspection, verification and extension. Scientists with an intention to simulate large or complex systems should seriously consider what it has to offer.

References

[1] For information on CCP5 see the project website at: http://www.ccp5.ac.uk
[2] W. Smith. Molecular dynamics on distributed (MIMD) parallel computers. *Theor. Chim. Acta*, **84**, 385 (1993).
[3] W. Smith. Molecular dynamics on hypercube parallel computers. *Comput. Phys. Comm.*, **62**, 229 (1991).
[4] A.R.C. Raine, D. Fincham, W. Smith. Systolic loop methods for molecular dynamics using multiple transputers. *Comp. Phys. Comm.*, **55**, 13 (1989).
[5] M.R.S. Pinches, D. Tildesley, W. Smith. Large scale molecular dynamics on parallel computers using the link cell algorithm. *Mol. Simul.*, **6**, 51 (1991).
[6] D.C. Rapaport. Multi-million particle molecular dynamics. II. Design considerations for distributed processing. *Comput. Phys. Comm.*, **62**, 217 (1991).
[7] S.J. Weiner, P.A. Kollman, D.T. Nguyen, D.A. Case. An all atom force field for simulations of proteins and nucleic acids. *J. Comp. Chem.*, **7**, 230 (1986).
[8] W.F. van Gunsteren, H.J.C. Berendsen. *Groningen Molecular Simulation (GROMOS) Library Manual*, BIOMOS, Nijenborgh, 9747 Ag Groningen, The Netherlands (1987).
[9] A.D. MacKerell Jr., B. Brooks, C.L. Brooks III, L. Nilsson, B. Roux, Y. Won, M. Karplus. CHARMM: the energy function and its parameterization with an overview of the program. *The Encyclopedia of Computational Chemistry*, Vol. 1, P.v.R. Schleyer, N.L. Allinger, T. Clark, J. Gasteiger, P.A. Kollman, H.F. Schaefer.
[10] B.G. Dick, A.W. Overhauser. Theory of dielectric constants of alkali halide crystals. *Phys. Rev. B*, **112**, 90 (1958).
[11] D. Fincham, P.J. Mitchell. Shell model simulations by adiabatic dynamics. *J. Phys. Condens. Matter*, **5**, 1031 (1993).
[12] P.J.D. Lindan, M.J. Gillan. Shell-model molecular-dynamics simulation of superionic conduction in CAF_2. *J. Phys. Condens. Matter*, **5**, 1019 (1993).
[13] M.W. Finnis, J.E. Sinclair. A simple empirical N body potential for transition metals. *Philos. Mag. A*, **50**, 45 (1984).
[14] J. Tersoff. Modelling solid state chemistry: interaction potentials for multicomponent systems. *Phys. Rev. B*, **39**, 5566 (1989).
[15] S.L. Mayo, B.D. Olafson, W.A. Goddard. DREIDING: a generic force field for molecular simulations. *J. Phys. Chem.*, **94**, 8897 (1990).
[16] M.P. Allen, D.J. Tildesley. *Computer Simulation of Liquids*, Clarendon Press, Oxford (1989).
[17] D.J. Evans, G.P. Morriss. Non-Newtonian molecular dynamics. *Comput. Phys. Rep.*, **1**, 297 (1984).
[18] W.G. Hoover. Canonical dynamics: equilibrium phase-space distributions. *Phys. Rev. A*, **31**, 1695 (1985).
[19] S. Melchionna, G. Ciccotti, B. Holian. Hoover NPT dynamics for systems varying in shape and size. *Mol. Phys.*, **78**, 533 (1993).
[20] H.J.C. Berendsen, J.P.M. Postma, W.F. van Gunsteren, A. DiNola, J.R. Haak. Molecular dynamics with coupling to an external bath. *J. Chem. Phys.*, **81**, 3684 (1984).
[21] J.P. Ryckaert, G. Ciccotti, H.J.C. Berendsen. Numerical integration of the Cartesian equations of motion of a system with constraints: molecular dynamics of n-alkanes. *J. Comput. Phys.*, **23**, 327 (1977).
[22] H.C. Andersen. Rattle: a velocity version of the SHAKE algorithm for molecular dynamics calculations. *J. Comput. Phys.*, **52**, 24 (1983).
[23] H. Goldstein. *Classical Mechanics*, Addison Wesley (1980).
[24] D.J. Evans. On the representation of orientation space. *Mol. Phys.*, **34**, 317 (1977).
[25] D. Fincham. Leapfrog rotational algorithms. *Mol. Simul.*, **8**, 165 (1992).
[26] T.F. Miller, M. Eleftheriou, P. Pattnaik, A. Ndirango, D. Newns, G.J. Martyna. Symplectic quaternion scheme for biophysical molecular dynamics. *J. Chem. Phys.*, **116**, 8649 (2002).
[27] T.R. Forester, W. Smith. Shake, rattle and roll: efficient constraint algorithms for linked rigid bodies. *J. Comput. Chem.*, **19**, 102 (1998).
[28] S. Brode, R. Ahlrichs. An optimised MD program for a vector computer cyber 205. *Comput. Phys. Commun.*, **42**, 41 (1986).
[29] R.W. Hockney, J.W. Eastwood. *Computer Simulation Using Particles*, McGraw-Hill International (1981).
[30] C. Kittel. *Solid State Physics*, John Wiley and Sons (1986).
[31] I.T. Todorov, W. Smith, K. Trachenko, M.T. Dove. DL_POLY_3: new dimensions in molecular dynamics simulations via massive parallelism. *J. Mater. Chem.*, **16**, 1911 (2006).
[32] W. Smith. A replicated data molecular dynamics strategy for the parallel Ewald sum. *Comput. Phys. Commun.*, **67**, 392 (1992).
[33] J. Hautman, J., M.L. Klein. An Ewald summation method for planar surfaces and interfaces. *Mol. Phys.*, **75**, 379 (1992).
[34] U. Essmann, L. Perera, M.L. Berkowitz, T. Darden, H. Lee, L.G. Pedersen. A smooth particle mesh Ewald method. *J. Chem. Phys.*, **103**, 8577 (1995).
[35] I.J. Bush. *The Daresbury Advanced Fourier Transform*, Daresbury Laboratory (1999).
[36] I.J. Bush, I.T. Todorov, W. Smith. A DAFT DL_POLY distributed memory adaptation of the smoothed particle mesh Ewald method. Accepted for publication, *Comput. Phys. Commun.* (2006).
[37] K. Trachenko, M.T. Dove, E. Artacho, I.T. Todorov, W. Smith. Atomistic simulations of resistance to amorphization by radiation damage. *Phys. Rev. B*, **73**, 174207 (2006).
[38] W. Smith, T.R. Forester. Parallel Macromolecular Simulations and the replicated data strategy II: the RD-SHAKE algorithm. *Comput. Phys. Comm.*, **79**, 63 (1993).

[39] W. Smith, I.T. Todorov, T.R. Forester, M. Leslie. The DL_POLY_2 user manual. Daresbury Laboratory (2006) available from the DL_POLY website[43].

[40] I.T. Todorov, W. Smith. The DL_POLY_3 user manual. (2006), Daresbury Laboratory (2006) available from the DL_POLY website[43].

[41] The CygWin Linux API emulator is available from: http://sources.redhat.com/cygwin/

[42] W. Smith. The DL_POLY java graphical user interface II. Daresbury Laboratory (2003) available from the DL_POLY website[44] (2003).

[43] The DL_POLY website is located at: http://www.ccp5.ac.uk/DL_POLY/